LINC COMPUTER USER-INTERACTIVE PROGRAMS AND MACRO INSTRUCTIONS

Walter E. Reynolds
Robert B. Tucker
Timothy B. Coburn
James C. Bridges

Technical Report No. IRL-1055

May 1, 1967

Prepared under:

National Aeronautics and Space Administration
Grant NsG 81-60
National Institutes of Health
Grant FR 00151
Air Force Office of Scientific Research
Contract AF 49(638)-1599

Principal Investigator: J. Lederberg
Director, Instrumentation Research Laboratory: E. Levinthal

Instrumentation Research Laboratory, Department of Genetics
Stanford University School of Medicine
Palo Alto, California

ABSTRACT

This report describes four program packages for use on the LINC computer.

(1) A program package which enables the LINC and a Teletype to be used as a very sophisticated desk calculator including graphical output with a Calcomp plotter.

(2) A general purpose double precision floating point subroutine package for the LINC.

(3) A set of input-output routines providing for the communication of octal, decimal and alphanumeric information via a Teletype.

(4) Also included is additional information on the LOSS system (see "An Operating System for the LINC Computer," R. K. Moore, NASA Technical Report No. IRL-1038) under which the above packages may be used.

TABLE OF CONTENTS

I. INTRODUCTION

During the last two years the LINC Programming Group of the Genetics Department, Stanford University School of Medicine, has written numerous LINC programs for use by the biological researchers of the department. Heavy emphasis has been placed upon providing for conversational input-output between the user and the computer. Thus the researcher, by means of a Teletype, can interact with his program and any instrumentation which may be on line with the computer. The three program packages described in this report have been particularly useful in handling the above applications. It is because of their flexibility and usefulness that they are herein reported for the use of other interested LINC users.

As with any piece of computer software, these packages are somewhat dependent upon the configuration of the input-output interfaces. The pertinent aspects of these interfaces are described in detail in the appendices of the report. A prospective user is urged to check his configuration against that given. If a difference exists it usually can be accommodated by judicious changes of a few instructions (OPR's, SXL's, ATR's, etc.) in the programs. The likely places where changes may be needed are indicated in the individual program descriptions.

The first program described, CALCULATOR III, is a complete program that enables the LINC and a Teletype to perform in a manner quite comparable to the most sophisticated electronic calculators on the market today. In addition, vector or single dimension array operations are included, direct communication with data blocks on LINC tape is permitted, and if a Calcomp plotter is available, output may be graphically displayed.

The second package is a set of floating-point routines. They also exist in CALCULATOR III, but here in a form more suitable for inclusion in any LINC program where double-precision floating-point arithmetic is desired. They occupy two quarters of LINC memory and

when so included, become a comprehensive set of floating point macro instructions.

The third package contains numerous general purpose routines in source code form invaluable to any LINC program where conversational input-output is desired. These may be inserted into LINC programs as desired to allow octal, decimal or alphanumeric communication with the LINC using a Model 33 Teletype in half-duplex mode.

These packages are presently utilized under the LOSS system, a general description of which is contained in Section V of this report.

Magnetic tape copies of CALCULATOR III, the floating point routines, and the source coding (in LOSS) of the input-output routines will be supplied upon request to any interested user. These will be placed upon a LINC tape along with the LOSS monitor (see Section V).

Users wishing such a copy are requested to send a marked LINC tape with their request to:

LINC Programming Group
Instrumentation Research Laboratory
Genetics Department
Stanford University School of Medicine
Palo Alto, California 94304

II. CALCULATOR III

The calculator program was designed and written to allow use of the LINC for calculations and simple statistics without special programming for each user. Calculator I contained all the features of a desk calculator plus some basic functions; sine, cosine, square root, etc. With the addition in 1964 of 1024 words of memory, Calculator I was expanded to handle lists of data (one-dimensional arrays) and renamed Calculator II. In 1966, Calculator II was rewritten to incorporate more complicated statistical routines and more extensive input-output functions; this is Calculator III. At present, new routines are still being added but no changes in the structure of the program are foreseen.

Calculator III is organized in a fashion acceptable to non-computer personnel. It has been found that after 10 minutes of instruction, the program can be used by persons with no previous knowledge of the LINC or digital computers. More efficient use can be made by more experienced operators. Due to the variety of users, the following users guide may be either too explicit or too general. Additional documentation is available.

A variety of operations are available in the calculator. These are listed below for reference.

	A	Add	O	Reverse
	S	Subtract	I	Set Pointer
	D	Divide	N	Print Fixed
	M	Multiply	P	Print Floating
STACK	E	Exponential	Q	Square Root
	L	Log (natural)	R	Read
	K	Arctan	W	Write
	G	Sine	X	Type Pointer
	H	Cosine	Z	Chi Square

CALCOMP	CL	Plot Line
	CP	Plot Point
BLOCK	BA	Block Add
	BS	Block Subtract
	BM	Block Multiply
	BD	Block Divide
	BR	Block Read
	BW	Block Write
VECTOR	VA	Vector Add
	VS	Vector Subtract
	VD	Vector Divide
	VM	Vector Multiply
	VE	Vector Exponential
	VL	Vector Log
	VK	Vector Arctan
	VG	Vector Sine
	VH	Vector Cosine
	VI	Fill Vector
	VN	Vector Print Fixed
	VP	Vector Print Floating
	VQ	Vector Square Root
	VU	Mean and Standard Deviation
	VT	"Student's" T-Test
	VF	Sum
	VJ	Product

The methods for using these operations are described in the following pages. The method for input of data (not mentioned above) involves the use of a pointer and stack. Hence, it seems worthwhile to describe first the pertinent features of memory allocation and data storage.

1. Data Storage and Memory Allocation

A. The upper half of memory is used for data. Location 2000 through 3734 (1734_8 12 bit words) function as a stack or list. Since double precision floating point arithmetic is used (three 12 bit words per stack cell), there are 329 stack cells available. These are numbered from 0 to 328. Stack cell 1 corresponds to location 2003, 2004, and 2005; and so forth.

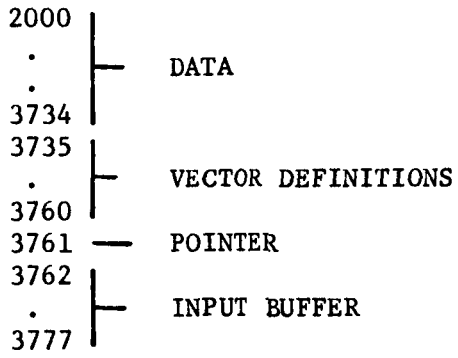
The addressing of the stack cells is handled by a pointer for the operations listed as "stack operations" on the previous page. For example: set the pointer to stack cell 17 by command "I18;". Type in an integer "349;". The pointer increments itself and directs the integer into stack cell 18. The next integer, "638;", will be placed in stack cell 19.

The pointer location is 3761. This 12 bit word contains the number of the stack cell at which the pointer is looking. In the above example it would contain 18. Locations 3762 through 3777 are used as an input buffer. All characters typed in are stored here until the semicolon is struck.

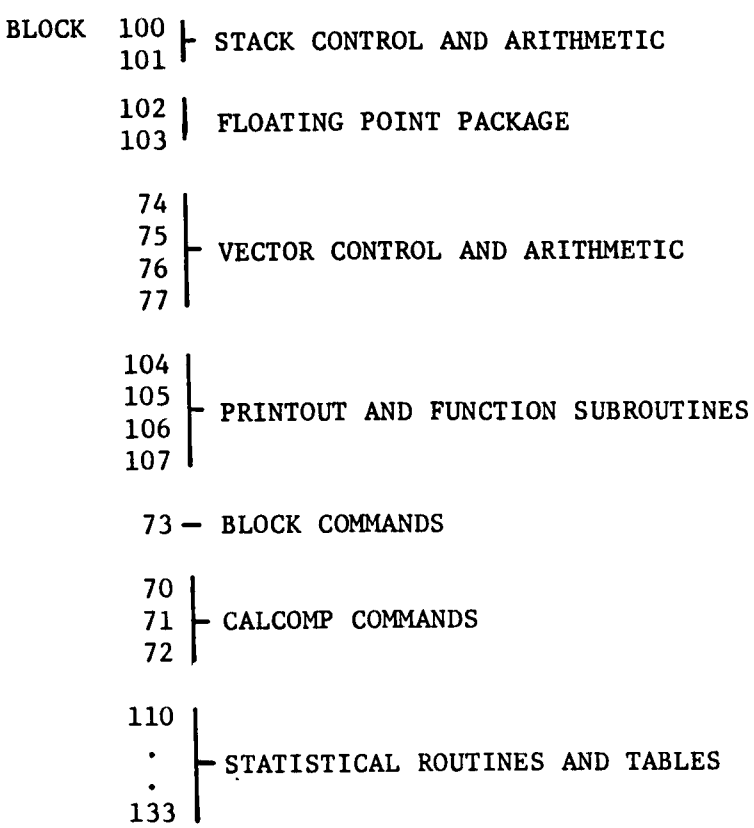
Locations 3735 through 3760 contain vector definitions. Vectors are merely lists within the stack. They can be defined by the user and subsequently used by number. For example: a vector may be defined as a list starting in stack cell 20 and consisting of the 50 consecutive stack cells. The use of these "vectors" will be explained in Section 3.

B. The lower half of memory is used for the program. In general, the first two quarters, 0 and 1, are used for the various control routines, while quarters 2 and 3 contain the floating point routines. Since these latter quarters are used by almost all the control routines, they are always present in core.

C. Diagram of Upper Half of Memory



D. Diagram of Tape Storage



2. Stack Operations

The eighteen operations listed in this group have one consistent similarity; they are called with a single letter. In addition, most of them refer to the pointer for their data values, and most of them carry out one operation per command.

A. Pointer Control

Ibbb;	Set the pointer for input into bbb. This actually sets it at bbb-1.
x;	Type the value of the pointer.

B. Data Input

+329.56;	Increment pointer and store 3.2956×10^2 in the appropriate stack cell.

-1.5,-99; Increment pointer and store -1.5×10^{-99} in the stack cell.

7; Most data looks like this.

NOTE: Spaces are not allowed within a command. Errors, when detected, cause immediate exit from the requested routine and NIX is typed out.

Exponents are allowed from 10^{-99} to 10^{+99}. Larger exponents are not typed out correctly although they are correct in memory.

Eight significant digits can be stored in 24 bits. Typing in more than 8 is meaningless.

C. Data Output

Nbbb; Type in fixed point a four digit number found in cell bbb.

Pbbb; Type in floating point form the six most significant digits and the exponent of cell bbb.

These two operations may be strung together in the form P1,2,3,4,100; "Print stack cells 1,2,3,4, and 100".

D. Arithmetic Routines

X_n Represents the stack cell with the pointer.

X_{n-1} Represents the stack cell preceding the pointer.

$\longrightarrow$ Indicates that the left side replaces the right.

A; $X_{n-1} + X_n \longrightarrow X_{n-1}$ Pointer decremented to X_{n-1}

S; $X_{n-1} - X_n \longrightarrow X_{n-1}$ Pointer decremented to X_{n-1}

M; $(X_{n-1}) \cdot (X_n) \longrightarrow X_{n-1}$ Pointer decremented to X_{n-1}

D; $X_{n-1} / X_n \longrightarrow X_{n-1}$ Pointer decremented to X_{n-1}

Note the position of the operands in the subtract and divide. Because these operations are not symmetrical the following operation has been included:

O; $X_{n-1} \rightleftarrows X_n$ i.e., reverse the last two locations.

The arithmetic operations perform as a "fold down stack." See examples at end of Section II.

E. Functions

L;	$\log_e X_n \rightarrow X_n$	Pointer unchanged
E;	$e^{Xn} \rightarrow Xn$	Pointer unchanged
Q;	$X_n \rightarrow X_n$	Pointer unchanged
G;	$\text{Sin } X_n \rightarrow X_n$	Pointer unchanged
H;	$\text{Cos } X_n \rightarrow X_n$	Pointer changed
K;	$\text{Arctan } X_n \rightarrow X_n$	Pointer unchanged

F. Miscellaneous

W;	Write upper memory into blocks 114-117.
R;	Read upper memory from blocks 114-117.
Zm,n;	See statistical routines-Chi square.

G. Examples of Operations Using the Stack

(1) $\left(\frac{1.1 + 35.7}{2.3} - 1\right) \times 22.4$ Evaluate

I0;	Set pointer to cell-1
1.1;	Put 1.1 into cell 0
35.7;	Put 35.7 into cell 1
A;	Add and store in cell 0

2.3;	Put 2.3 into cell 1
D;	Divide into previous sum and store in cell 0
1;	Put 1 into cell 1
S;	Subtract from previous quotient store cell 0
22.4;	Put 22.4 into cell 1
M;	Multiply times cell 0 and store
P;	Print the value at the pointer. (PQ would have the same result since the pointer is at cell 0.)

(2) Evaluate $\frac{1}{(1x2x3x4x5x6)}$

I20;	Set pointer to cell 19
1;	Put 1 in cell 20
2;	Put 2 in cell 21
3;	Put 3 in cell 22
4;	Put 4 in cell 23
5;	Put 5 in cell 24
6;	Put 6 in cell 25
m;	Multiply 6 x 5 store in cell 24
m;	Multiply 30 x 4 store in cell 23
m;	Multiply 120 x 3 store in cell 22
m;	Multiply 360 x 2 store in cell 21
m;	Multiply 770 x 1 store in cell 20
I;	Put 1 in cell 21
O;	Reverse cell 21 and 20
D;	Divide 1 by 720 store in cell 20
P;	Print result (P20 would also print the result)

3. Vector Operations

A vector is a one-dimensional array or list with a beginning point and a length. In terms of the stack concept, a vector is a defined subset of the stack. That is, the stack might be defined as a vector

starting at 0 with a length of 329 cells. Another vector might start at cell 20 with a length of 10 cells, and so forth. The use of these vectors will become more apparent in the examples below. The following operations can be performed with vectors.

A. Defining a Vector

The user is allowed to define the vectors he chooses to use. For simplicity vector names are the digits 0 to 9.

V1, 10, 50:	Define vector 1, starting in cell 10 with length of 50 cells, i.e., cells 10-59.
V9, 2, 20:	Define vector 9, starting in cell 2 with length of 20 cells, i.e., cells 2-21.

The fact that vector 1 and 9 are overlapping is inconsequential.

B. Vector Arithmetic

VA1, 2, 3:	Add the first cell of vector 1 to the first cell of vector 2, store in the first cell of vector 3 and continue with each succeeding element.
VS1, 2, 3:	Subtract vector 2 from vector 1, store in 3.
VM1, 2, 3:	Multiply vector 1 by 2, store in 3.
VD1, 2, 3:	Divide 1 by 2, store in 3.

(1) If vector 2 (or 1) is shorter than 1, the operation will loop back to the beginning. The longer vector must be satisfied before the operation terminates.

(2) If the storage vector (vector 3) is too short an error message is typed out.

(3) If less than three vectors are specified in an operation, the last one or two will be assumed.

VM3:	Multiply 3 by 3, store in 3, i.e., square it.

C. Vector Functions

VQ1, 3: Square root of each element in vector 1 is stored in vector 3.

VE, VL, VG, VH, VK are similar.

D. Vector Output

VP1, 4, 5: Print vectors 1, 4 and 5 in three columns.

VN5, 4, 1: Print fixed 5, 4, and 1 in three columns.

E. Miscellaneous

VI3: Fill vector 3 with an arithmetic series; the starting number is in the first cell of vector 3; the increment in second cell.

VF1, bbb: Compute the sum of the cells in vector 1, store the result in cell bbb.

VJ1, bbb: Compute the product of the cells in vector 1, store result in cell bbb.

(1) Sense switch 0 causes exit from the print routines.

(2) When printing in floating form, a maximum of 5 vectors can be printed since this fills the teletype page. The corresponding maximum for the fixed form is 9.

(3) Vectors must be defined before they are used. The calculator prints NIX if an undefined vector is requested. However, since vectors remain defined after use, it is more likely that one forgets to redefine a vector. This may or may not cause an error message. The calculator doesn't know if you meant to redefine a vector except when it is too small for storage.

F. Examples Using Vectors

(1) Evaluate $3.5\sqrt{\frac{1+e^{X}}{X^{2}-4.62}}$ for X from 1 to 100

```
V1, 0, 100;        Define vector 1.
I0;                Set pointer to -1.
1;                 Put 1 in cell 0.
1;                 Put 1 in cell 1.
VI1;               Fill vector 1 with the digits 1-100.
V2, 100, 100;      Define vector 2.
VM1, 1, 2;         Square vector 1, store in 2.
VE1;               Exponential of vector 1, store in 1.
I200;              Set pointer to 199.
1;                 Put 1 in cell 200.
V3, 200, 1;        Define vector 3.
VA1, 2, 1;         Add the value 1 to vector 1, store in 1.
I200;              Set pointer to 199.
4.62;              Put 4.62 in 200.
VS2, 3, 2;         Subtract from vector 2, store in 2.
VD1, 2;            Divide vector 1 by 2, store in 2.
VQ2;               Square root of 2.
I200;              Set pointer to 199.
3.5;               Put 3.5 in 200.
VM3, 2;            Multiply times vector 2.
VP2;               Print vector 2.
```

(2) Evaluate $\sqrt{\frac{\Sigma(x^2 - y)}{\sin(3.6-x)}}$ Where x and y each have 50 values to be typed in.

```
I0;                Set pointer to 0.
(X1);
(X2);              Input X values.
(Y1);
(Y2);              Input Y values.
W;                 Write on tape in case of blunder.
V1,0,50,2,50,50,3,100,50;    Define vectors 1, 2, and 3.
I150;              Set pointer to 149.
3.6;               Put 3.6 in cell 150.
```

V4, 150, 1;	Define vector 4.
VS1, 4, 3;	Subtract vector 1 from 4, store in 3.
VG3;	Sine of vector 3.
VM1;	Square vector 1.
VS1, 2;	Subtract vector 2 from 1, store in 2.
VD3,2;	Divide vector 3 by 2, store in 2.
VF2, 300;	Sum vector 2, store in cell 300.
I301;	Set pointer to 300.
Q;	Square root.
P; or P300;	Print.
(3) Evaluate	$35.5 - \left(\frac{1}{4 + 2.3}\right)^{1/2} \times e^{-1.9}$
I100;	Set pointer at 99.
4;	Put 4 in 100.
2.3;	Put 2.3 in 101.
A;	Add and store in 100.
1;	Put 1 in 101.
0;	Reverse.
D;	Divide 1 by 6.3, store in 100.
Q;	Square root of value in 100.
-1.9;	Put -1.9 in 101.
e;	Exponential of 101.
M;	Multiply times 100, store in 100.
35.5;	Put 35.5 in 101.
0;	Reverse.
S;	Subtract 101 from 100, store in 100.
P;	Print result. (P100 would also print the result.)

From the examples above, it should be clear that the pointer takes care of itself in most cases. It obeys a fairly logical set of rules, which, with a little practice can be relegated to a subconscious corner of the human memory.

Subtotals may be printed out at any time with no effect on the data stores in those cells. Data is not destroyed by re-reading the system. Hence, one is free to try any configuration of commands which seem logical. At worst they will cause the program to halt. The read on tape (R) and write on tape (W) instructions are for the preservation of long data lists. If one fears that an operation may alter irreparably the data they have arduously typed in, W; will keep all data, the pointer, and the vector definitions on tape. After executing R; the calculator system will be in precisely the same configuration as it was before the last W.

4. Block Operations

The block operations provide input-output between the calculator and Linc tape. They operate on stack cells 0-255 for both input and output.

BRd,x;	Read block x, unit d into quarter 1; convert each 12 bit word to double precision floating point and store sequentially in stack cells 0 through 255.
BWd,x;	Fix each double precision floating point word in cells 0-255, write on tape block x, unit d.
*BAd,x_1,x_2,x_3;	Read block x_1, unit d into quarter 1; add each value to the corresponding stack cell; continue for x_2, x_3, etc.
*BSd,x;	Subtract block x, unit d from stack cells 0-255.
*BMd,x;	Multiply block x, unit d times stack cells 0-255.
*BDd,x;	Divide block x, unit d into stack cells 0-255.

*More than one block may be specified, as in the above block add. However, the limit on the number of characters in a teletype command is 28. Therefore, the following command represents the maximum length

which will be correctly interpreted: BA1, 102, 103, 104, 105, 106, 107;.

5. Calcomp Routines

At present, the Calcomp routines provide a quick method of dumping a string of data points. I have used them extensively for checking out other routines such as log and sine.

Form of Command:

C _ _,_,_,_; The "C" refers to Calcomp.

(1) The first parameter is either "L" or "P" indicating a connected line plot of an unconnected series of points.

(2) The second parameter designates the vector to be plotted. It must be an integer from 0 to 9.

(3) The third parameter designates the symbol, if any, for each point.

0	no symbol
1	0
2	□
3	△
4	◇
5	⬭
6	x
7	+
8	*

(4) The fourth parameter refers to the X dimension of the plot.

a. ,8, would produce a plot 8 inches long, or

b. ,I5, would produce a plot in which each point was spaced by 5 steps (100ths of an inch) on the X axis.

(5) The fifth parameter is exactly analogous to the fourth, but it refers to the Y dimension.

NOTE: By specifying an integer, as in a, the plot is independent of the absolute magnitude of the data. Specifying an incremented factor, as in b, makes the plot absolutely dependent on the magnitude of the data. At present, an integer can only specify inches, not parts of an inch.

6. Chi Square-N by N

There are many chi square tests. The method used by this program is taken from R. A. Fisher, Statistical Methods for Research Workers, Chapter IV, (see bibliography). The following excerpt from this chapter describes the use of this program.

"It should be noted that the methods employed in this chapter are not designed to measure the degree of association between one classification and another, but solely to test whether the observed departures from independence are or are not of a magnitude ascribable to chance."

Input

Data is entered starting in stack cell 0. It may be entered row by row, or column by column.

Execute Command

"Zn,m;" where n is the number of rows and m is the number of columns, assuming the data has been input row by row. Otherwise n is the number of columns and m the number of rows.

Output

Three values are output:

(1) Degrees of freedom

(2) Chi square

(3) Probability of independence associated with the above. The table used for computing the probability of independence is on the following page (Table 1).

n.	$P=.99$.	.98.	.95.	.90.	.80.	.70.
1	·000157	·000628	·00393	·0158	·0642	·148
2	·0201	·0404	·103	·211	·446	·713
3	·115	·185	·352	·584	1·005	1·424
4	·297	·429	·711	1·064	1·649	2·195
5	·554	·752	1·145	1·610	2·343	3·000
6	·872	1·134	1·635	2·204	3·070	3·828
7	1·239	1·564	2·167	2·833	3·822	4·671
8	1·646	2·032	2·733	3·490	4·594	5·527
9	2·088	2·532	3·325	4·168	5·380	6·393
10	2·558	3·059	3·940	4·865	6·179	7·267
11	3·053	3·609	4·575	5·578	6·989	8·148
12	3·571	4·178	5·226	6·304	7·807	9·034
13	4·107	4·765	5·892	7·042	8·634	9·926
14	4·660	5·368	6·571	7·790	9·467	10·821
15	5·229	5·985	7·261	8·547	10·307	11·721
16	5·812	6·614	7·962	9·312	11·152	12·624
17	6·408	7·255	8·672	10·085	12·002	13·531
18	7·015	7·906	9·390	10·865	12·857	14·440
19	7·633	8·567	10·117	11·651	13·716	15·352
20	8·260	9·237	10·851	12·443	14·578	16·266
21	8·897	9·915	11·591	13·240	15·445	17·182
22	9·542	10·600	12·338	14·041	16·314	18·101
23	10·196	11·293	13·091	14·848	17·187	19·021
24	10·856	11·992	13·848	15·659	18·062	19·943
25	11·524	12·697	14·611	16·473	18·940	20·867
26	12·198	13·409	15·379	17·292	19·820	21·792
27	12·879	14·125	16·151	18·114	20·703	22·719
28	13·565	14·847	16·928	18·939	21·588	23·647
29	14·256	15·574	17·708	19·768	22·475	24·577
30	14·953	16·306	18·493	20·599	23·364	25·508

·50.	·30.	·20.	·10.	·05.	·02.	·01.
·455	1·074	1·642	2·706	3·841	5·412	6·635
1·386	2·408	3·219	4·605	5·991	7·824	9·210
2·366	3·665	4·642	6·251	7·815	9·837	11·345
3·357	4·878	5·989	7·779	9·488	11·668	13·277
4·351	6·064	7·289	9·236	11·070	13·388	15·086
5·348	7·231	8·558	10·645	12·592	15·033	16·812
6·346	8·383	9·803	12·017	14·067	16·622	18·475
7·344	9·524	11·030	13·362	15·507	18·168	20·090
8·343	10·656	12·242	14·684	16·919	19·679	21·666
9·342	11·781	13·442	15·987	18·307	21·161	23·209
10·341	12·899	14·631	17·275	19·675	22·618	24·725
11·340	14·011	15·812	18·549	21·026	24·054	26·217
12·340	15·119	16·985	19·812	22·362	25·472	27·688
13·339	16·222	18·151	21·064	23·685	26·873	29·141
14·339	17·322	19·311	22·307	24·996	28·259	30·578
15·338	18·418	20·465	23·542	26·296	29·633	32·000
16·338	19·511	21·615	24·769	27·587	30·995	33·409
17·338	20·601	22·760	25·989	28·869	32·346	34·805
18·338	21·689	23·900	27·204	30·144	33·687	36·191
19·337	22·775	25·038	28·412	31·410	35·020	37·566
20·337	23·858	26·171	29·615	32·671	36·343	38·932
21·337	24·939	27·301	30·813	33·924	37·659	40·289
22·337	26·018	28·429	32·007	35·172	38·968	41·638
23·337	27·096	29·553	33·196	36·415	40·270	42·980
24·337	28·172	30·675	34·382	37·652	41·566	44·314
25·336	29·246	31·795	35·563	38·885	42·856	45·642
26·336	30·319	32·912	36·741	40·113	44·140	46·963
27·336	31·391	34·027	37·916	41·337	45·419	48·278
28·336	32·461	35·139	39·087	42·557	46·693	49·588
29·336	33·530	36·250	40·256	43·773	47·962	50·892

TABLE 1

7. Mean and Standard Deviation

Example of use:

"VU 1, 300, 301;"

The above command computes the mean and standard deviation of the values in vector 1. Leave mean in location 300, standard deviation in 301.

$$\text{Mean} = \frac{\Sigma x}{n}$$

$$\text{Standard Deviation} = \frac{n\Sigma x^2-(\Sigma x)^2}{n(n-1)}$$

8. "Student's"(Fisher's) T-Test

Example of use:

"VT 1, 2;"

Using the values found in vectors 1 and 2, the routine computes the appropriate T value (see next page), searches the probability table (Table 2) for the corresponding P value, and types out T, P and N.

The Student's T-Test is a measure of the significance between two means. It is designed to take small populations into account, although it works equally well for large populations.

The probability table consists of a stored matrix 28 by 12. If N is 28 or greater, it is assumed to be infinity. Smaller values of N each have 12 entries in the table. There is a slight discontinuity between N of 27 and 28.

A large P value, i.e., a value approaching 1 indicates that the means of the two populations are similar. A small P value, .05 or less, represents a 95 percent probability that the means are different.

The following explanation is taken from Fisher's Statistical Methods for Research Workers, page 122 (see bibliography).

Comparison of Two Means

"In experimental work it is even more frequently necessary to test whether two samples differ significantly in their means, or whether they may be regarded as belonging to the same population. In the latter case any difference in treatment which they may have received will have shown no significant effect.

If $x_1, x_2, \ldots, x_{n_1} + I$ and $x'_1, x'_2, \ldots, x'_{n_2} + I$ be two samples, the significance of the difference between their means may be tested by calculating the following statistics:

$$\bar{x} = \frac{I}{n_1 + I} S(x), \quad \bar{x}' = \frac{I}{n_2 + I} S(x'),$$

$$s^2 = \frac{I}{n_1 + n_2} S(x - \bar{x})^2 + S(x' - \bar{x}')^2$$

$$t = \frac{\bar{x} - \bar{x}'}{s} \quad \frac{(n_1 + I)(n_2 + I)}{n_1 + n_2 + 2}$$

$$n = n_1 + n_2$$

The means are calculated as usual; the standard deviation is estimated by pooling the sums of squares from the two samples and dividing by the total number of the degrees of freedom contributed by them; if σ were the true standard deviation, the variance of the first mean would be $\sigma^2/(n_1 + I)$, of the second mean $\sigma^2/(n_2 + I)$, and therefore that of the difference would be $\sigma^2\, I/(n_1 + I) + I/(n_2 + I)$; t is therefore found by dividing $\bar{x} - \bar{x}'$ by its standard error as estimated, and the error of the estimation is allowed for by entering the table with n equal to the number of degrees of freedom available

for estimating s; that is $n = n_1 + n_2$. It is thus possible to extend "Student's" treatment of the error of a mean to the comparison of the means of two samples."

n.	$P=.9$	.8.	.7.	.6.	.5.	.4.	.3.	.2.	.1.	.05.	.02.	.01.
1	.158	.325	.510	.727	1.000	1.376	1.963	3.078	6.314	12.706	31.821	63.657
2	.142	.289	.445	.617	.816	1.061	1.386	1.886	2.920	4.303	6.965	9.925
3	.137	.277	.424	.584	.765	.978	1.250	1.638	2.353	3.182	4.541	5.841
4	.134	.271	.414	.569	.741	.941	1.190	1.533	2.132	2.776	3.747	4.604
5	.132	.267	.408	.559	.727	.920	1.156	1.476	2.015	2.571	3.365	4.032
6	.131	.265	.404	.553	.718	.906	1.134	1.440	1.943	2.447	3.143	3.707
7	.130	.263	.402	.549	.711	.896	1.119	1.415	1.895	2.365	2.998	3.499
8	.130	.262	.399	.546	.706	.889	1.108	1.397	1.860	2.306	2.896	3.355
9	.129	.261	.398	.543	.703	.883	1.100	1.383	1.833	2.262	2.821	3.250
10	.129	.260	.397	.542	.700	.879	1.093	1.372	1.812	2.228	2.764	3.169
11	.129	.260	.396	.540	.697	.876	1.088	1.363	1.796	2.201	2.718	3.106
12	.128	.259	.395	.539	.695	.873	1.083	1.356	1.782	2.179	2.681	3.055
13	.128	.259	.394	.538	.694	.870	1.079	1.350	1.771	2.160	2.650	3.012
14	.128	.258	.393	.537	.692	.868	1.076	1.345	1.761	2.145	2.624	2.977
15	.128	.258	.393	.536	.691	.866	1.074	1.341	1.753	2.131	2.602	2.947
16	.128	.258	.392	.535	.690	.865	1.071	1.337	1.746	2.120	2.583	2.921
17	.128	.257	.392	.534	.689	.863	1.069	1.333	1.740	2.110	2.567	2.898
18	.127	.257	.392	.534	.688	.862	1.067	1.330	1.734	2.101	2.552	2.878
19	.127	.257	.391	.533	.688	.861	1.066	1.328	1.729	2.093	2.539	2.861
20	.127	.257	.391	.533	.687	.860	1.064	1.325	1.725	2.086	2.528	2.845
21	.127	.257	.391	.532	.686	.859	1.063	1.323	1.721	2.080	2.518	2.831
22	.127	.256	.390	.532	.686	.858	1.061	1.321	1.717	2.074	2.508	2.819
23	.127	.256	.390	.532	.685	.858	1.060	1.319	1.714	2.069	2.500	2.807
24	.127	.256	.390	.531	.685	.857	1.059	1.318	1.711	2.064	2.492	2.797
25	.127	.256	.390	.531	.684	.856	1.058	1.316	1.708	2.060	2.485	2.787
26	.127	.256	.390	.531	.684	.856	1.058	1.315	1.706	2.056	2.479	2.779
27	.127	.256	.389	.531	.684	.855	1.057	1.314	1.703	2.052	2.473	2.771
28	.127	.256	.389	.530	.683	.855	1.056	1.313	1.701	2.048	2.467	2.763
29	.127	.256	.389	.530	.683	.854	1.055	1.311	1.699	2.045	2.462	2.756
30	.127	.256	.389	.530	.683	.854	1.055	1.310	1.697	2.042	2.457	2.750
∞	.12566	.25335	.38532	.52440	.67449	.84162	1.03643	1.28155	1.64485	1.95996	2.32634	2.57582

TABLE 2

III. FLOATING POINT ROUTINES

1. Background

The present (1966) version of the floating point routines improves on the past (1965) version in speed, accuracy, and operations. The additional features which have been added are:

a. A round off procedure (used internally).

b. Inclusion of subtract integer and square root operations.

c. Direct entry, i.e., jump 1000.

The only incompatibility with the previous (1965) version is contained in the direct entry. This can be remedied by either eliminating the three instructions entry required in programs using the old version or by merely changing the first four instructions in the present floating point routines to: ADD 1445, STC 17, CLR, NOP.

Listing 2 at the end of this section shows the coding involved in the routines themselves.

2. Codes and Operations

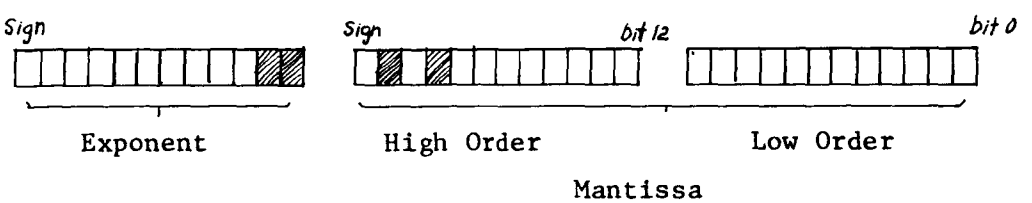
0.	Sqrt	Compute the square root of the value in operand. Leave in FAC.
1.	Cla	Clear and add (load) operand into FAC.
2.	Add	Add operand to FAC.
3.	Com	Complement operand; leave in FAC.
4.	Mul	Multiply operand by FAC; result in FAC.
5.	Fac/op	Divide the Fac by the operand; result in Fac.
6.	Op/Fac	Divide the operand by the Fac; result in Fac.
7.	I + Fac	Add an integer operand to the FAC.
10.	I x Fac	Multiply an integer operand by the Fac.
11.	Fac/I	Divide the Fac by an integer operand.
12.	I/Fac	Divide an integer operand by the Fac.
13.	Fix	Convert Fac to an integer; leave in LINC accumulator.

14.	Float	Convert an integer to a floating point word; leave in Fac.
15.	Clr	Clear Fac and operand.
16.	Max	Compare size of operand with Fac; larger left in Fac.
17.	Min	Compare size of operand with Fac; smaller left in Fac.
20.	Sgn	Check the sign of the operand; depending on whether it is positive,negative or zero, leave in the LINC accumulator +1, -1 or zero.
21.	Incr	Add Fac to operand and store in operand; i.e., add to memory.
22.	Sub	Subtract operand from Fac; leave result in Fac.
23.	Sto	Store Fac in operand; also, leave in Fac.
24.	Ssp	Set sign of operand positive; leave in Fac.
25.	Ssm	Set sign of operand minus; leave in Fac.
26.	Fac-I	Subtract integer operand from Fac. Result in Fac.

3. The Floating Point Format

The use of double precision floating point arithmetic seems essential if the LINC is to serve as a statistical processor. In using this type of arithmetic, the programmer trades speed and space for ease in dealing with large numbers. Programs which are extremely laborious to write and debug may become rather trivial using these routines (see mean and S.D. example).

A. Form of double precision floating point number:



The number above represents the integer 5 after it has been floated.

B. Floating an integer involves shifting the number right across the binary point until it is a fraction, and then counting the number of shifts to make up the exponent. In the above case, 5 = 101.000 in binary. Three shifts right produce 000.101. Since the binary point is always located between bits 23 and 22, the floating point number contains 101 in bits 22, 21 and 20. The exponent equals 3. Another way of representing this binary number is $.101 \times 2^3$.

C. Normalized floating point numbers always contain their most significant bit in bit 22. The above number could be represented in an unnormalized mode, such as $.010 \times 2^4$. But it is never represented this way in the floating point routines, since this would waste precision out at the right end. In its normalized mode, $.101 \times 2^3$, the number contains 23 bits of precision. This corresponds to more than 7 decimal digits.

D. Fixing a floating point number is the reverse of the float. It is shifted left across the binary point until the exponent equals zero. The fractional part remaining, if any, is either discarded or used for rounding off the integer.

E. Negative numbers are represented as the one's complement of positive numbers as in standard LINC integers. The mantissa (high and low order words) is merely complemented. Note that there is no sign bit in the low order word.

F. Negative exponents indicate that the number is less than one and has been shifted left until it is normalized. The sign of the exponent should not be confused with the sign of the mantissa. $.101 \times 2^{-3}$ is no more a negative number than is 5×10^{-3}. In the former the minus exponent indicates that if the number were fixed it would be .000101. The floating point routines would give a zero if requested to fix this number.

4. Method of Operation

A. The two quantities in a floating point operation, such as ADD, are called the Floating Accumulator and the Operand. The former

is maintained in locations 1120, 1121, and 1122. The latter is specified in the calling sequence.

B. Operations calling for an integer operand expect a standard LINC number, positive or negative. These operations are included strictly to save programming space. The integer is always floated before it is used.

C. The FIX operation leaves the integer value in the LINC accumulator. The last bit scaled off to the right is left in the LINC bit for round off purposes. Numbers larger than 3777 or smaller than .5 are fixed as 3777 and zero respectively. Negative numbers similarly out of range give -3777 and -0.

D. Errors such as zero divisors and negative square roots cause a jump to the end of the operation when they are discovered. The FAC may contain garbage at this point.

E. Index registers 12-17 are used by the floating point routines and are not restored at exit.

5. Timing

The timing for the various operations was computed by running through each operation 1000 times. Values were chosen which represented the worst case. For example: floating 1 requires some 4.5 milliseconds; whereas, floating 1000 requires about .5 milliseconds. The variation in the speed of the other operations is much less, about 1 or 2 milliseconds.

ADD	4.5 milliseconds
Float	4.5 milliseconds
Multiply	9 milliseconds
Divide	27 milliseconds
Fix	1 millisecond

6. Parameters and Use

A. Instruction Format

The format at the right represents a series of instructions inserted in any standard LINC program. Execution of floating point operations starts with the entry, Jmp 1000. Two parameters are required for each floating point operation following the entry. (See below for parameter description.) After the last parameter codes are again interpreted as standard LINC instructions.

```
.
.
.
Jmp 1000
 operand
 operation
 operand
 operation
 operand
 operation
.
.
.
```

B. The operand specification is an address.

(1) Direct address. The code 300, as an operand, is interpreted by the floating point routines as the address of the value required in the operation. If the operation requires an integer, only the value in location 300 is picked up. If the operation expects a floating point number the values in locations 300, 301 and 302 are picked up. They are assumed to be in the normalized floating point mode specified previously.

(2) Indirect address. The code 4300 refers to a value whose address is in location 300. The 4000 bit in the operand always indicates an indirect address. Any location may be used as an indirect address. This differs from the restriction in standard LINC codes of using only index registers for indirect addressing.

(3) Zero. A zero operand refers to the floating accumulator. Hence, in order to square a number in the FAC one need only specify a zero operand and the multiply operation, 4.

C. Operations

The operation codes with their descriptions are listed on page 22. The 4000 bit in the operation code indicates a continuing series (see example). When the 4000 bit is absent from an operation code the subsequent location will be executed as a standard LINC instruction.

There are 27 operations available in the floating point routines. Most of these are merely permutations of the ADD, MUL, DIV, FIX, and FLT. Used judiciously these permutations save a great deal of time and space in programming floating point operations.

The square root routine is included because it has been found quite useful.

D. Example: On the following page can be found a program for computing the mean and standard deviation of the numbers in locations 2000-3777 (listing 1). The formula used for this program is:

$$\text{Mean} = \frac{\Sigma x}{N} \quad , \quad \text{S.D.} = \sqrt{\frac{N\Sigma x^2 - (x)^2}{N(N-1)}} \quad .$$

Registers A and B, (locations 400-405) are used for storing respectively the sum and the sum of squares. They are also used for storing the results, the mean and standard deviation at the end.

```
TITLE---EXAMPLE;
 400
 401                    Register A
 402
 403
 404                    Register B
 405
 406 SETi1              Start
 407 2000
 410 JMP 1000
 411 400                Clear A
 412 4015
 413 403                Clear B
 414 15
 415 JMP 1000
 416 4001               Float Integer
 417 4014
 420 400                Add to Memory - Register A - Sum
 421 4021
 422 4001               Float Integer
 423 4014
 424 0                  Square
 425 4004
 426 403                Add to Memory - Register B - Sum of Squares
 427 21
 430 LDAi
 431 1
 432 ADM                Increment and Check for End
 433 2
 434 SAEi
 435 -3777
 436 JMP 415            Go to Next Integer
 437 JMP 1000
 440 407                Σx^2 . N (2000)
 441 4010
 442 403                Store in B
 443 4023
 444 400                Clear and Add Σx
 445 4001
 446 0                  Square
 447 4004
 450 0                  Complement
 451 4003
 452 403                Add to Memory - B
 453 4021
 454 407                Float N (2000)
 455 4014
 456 431                Subtract 1
 457 4026
 460 407                Times 2000
 461 4010
 462 403                Divide into B
 463 4006
 464 0                  Square Root
 465 4000
 466 403                Store in B - S.D.
 467 4023
 470 400                Clear and Add Σx
 471 4001
 472 407                Divide by N (2000)
 473 4011
 474 400                Store - Mean
 475 23
```

LISTING 1

```
TITLE--- FLT.
1000 CLR
1001 ADD 0
1002 ADD 1445
1003 STC 17
1004 SET↑13
1005 1121
1006 SET↑14
1007 1122
1010 SET↑15
1011 1124
1012 SET↑16
1013 1125
1014 LDA↑17
1015 AZE↑
1016 ADD 1032
1017 APO↑
1020 JMP 1025
1021 BCO↑
1022 -1777
1023 STC 1024
1024
1025 STA↑
1026 0
1027 STC 12
1030 LDA↑17
1031 ROL↑1
1032 ADA↑
1033 -367
1034 STC 1045
1035 ROL↑1
1036 STC 1047
1037 LDA 12
1040 STC 1123
1041 LDA↑12
1042 STC 1124
1043 LDA↑12
1044 STC 1125
1045
1046 SRO↑
1047 0
1050 JMP 1014
1051 STC 1045
1052 ADD 17
1053 ADA↑
1054 -1776
1055 STC 1057
1056 ADD 1045
1057
1060 CLR
1061 ADD 1123
1062 STA 15
1063 SCR 13
1064 STC 1125
1065 ADA↑
1066 13
1067 STC 1123
1070 ADD 0
1071 STC 1073
1072 JMP 1500
1073 HLT
1074 JMP 1532
1075 JMP 1072
1076 ADD 1120
1077 AZE
```

Pick up return address.

Set index registers to the floating accumulator (Fac), and to the argument (Arg.).

Pick up address of the operand.

Pick up and decode the operation code. Leave the result as a jump instruction.

Pick up the operand and put it into the argument register.

Jump to the appropriate operation.

Check for an exit from the floating point routines.

Compute return location and execute exit.

Float the argument.

LISTING 2

The Floating Point Package

```
1100 APO↑
1101 JMP 1105
1102 LDA 13
1103 SCR↑14
1104 JMP 1046
1105 ADD 1277
1106 APO↑
1107 JMP 1737
1110 COM
1111 ADD 1423
1112 STC 1116
1113 ADD 1122
1114 ROL↑1
1115 LDA 13
1116
1117 JMP 1046
1120
1121
1122
1123
1124
1125
1126
1127
1130
1131
1132
1133 SET 12
1134 0
1135 LDA 15
1136 BCO 13
1137 SCR↑14
1140 LDA 15
1141 LZE
1142 JMP 1174
1143 SCR↑14
1144 ADD 1120
1145 COM
1146 ADD 1123
1147 LZE
1150 COM
1151 AZE
1152 JMP 1174
1153 CLR
1154 JMP 1160
1155 HLT
1156 SET 12
1157 0
1160 LDA 13
1161 COM
1162 ADA 15
1163 AZE
1164 JMP 1174
1165 LDA 14
1166 STC 1172
1167 LDA 16
1170 COM
1171 LAM↑
1172 0
1173 ROR↑1
1174 APO↑
1175 XSK↑12
1176 CLR
1177 JMP 12
```

Fix the floating accumulator.

Floating Accumulator (Fac).

Argument (Arg).

B register.

Q (Quotient) register.

Compare the size of the Fac and the Arg. If the Fac is greater than or equal to the Arg., return to location p+1. Otherwise return to p+2.

LISTING 2 (cont.)

```
1200 CLR
1201 ADD 0
1202 STC 1252
1203 LDA 15
1204 AZE↑
1205 JMP 1252
1206 LDA 13
1207 AZE↑
1210 JMP 1251
1211 JMP 1474
1212 STC 1236
1213 ADD 1120
1214 COM
1215 ADD 1123
1216 AZE↑
1217 JMP 1233
1220 APO↑
1221 JMP 1224
1222 STC 12
1223 JMP 1227
1224 COM
1225 STC 12
1226 JMP 1730
1227 JMP 1517
1230 XSK↑12
1231 JMP 1227
1232 JMP 1650
1233 JMP 1604
1234 JMP 1474
1235 SRO↑
1236 0
1237 JMP 1250
1240 APO↑
1241 JMP 1250
1242 LDA 13
1243 ROL↑1
1244 LDA 15
1245 ROR↑1
1246 JMP 1522
1247 JMP 1650
1250 JMP 1070
1251 JMP 1557
1252
1253 JMP 1474
1254 STC 1674
1255 JMP 1632
1256 JMP 1550
1257 JMP 1625
1260 ADD 1120
1261 ADD 1126
1262 STC 1126
1263 SET↑12
1264 -14
1265 CLR
1266 SRO
1267 1130
1270 JMP 1604
1271 LDA 15
1272 ROR↑1
1273 JMP 1522
1274 XSK↑12
1275 JMP 1265
1276 SET↑12
1277 -13
```

Add the Arg to the Fac. Leave the result in the Fac.

Multiply the Arg times the Fac. Leave the result in the Fac.

LISTING 2 (cont.)

```
1300 CLR
1301 SRO
1302 1127
1303 JMP 1604
1304 LDA 15
1305 ROR↑1
1306 JMP 1522
1307 XSK↑12
1310 JMP 1300
1311 LDA
1312 1126
1313 STC 1123
1314 JMP 1650
1315 JMP 1070
1316 JMP 1673
1317
1320 JMP 1474
1321 STC 1674
1322 JMP 1632
1323 JMP 1505
1324 JMP 1046
1325 JMP 1156
1326 JMP 1331
1327 JMP 1550
1330 JMP 1335
1331 JMP 1730
1332 JMP 1517
1333 JMP 1650
1334 JMP 1730
1335 JMP 1616
1336 ADD 1123
1337 COM
1340 ADD 1120
1341 STC 1126
1342 STC 1131
1343 STC 1132
1344 SET↑12
1345 1130
1346 XSK↑12
1347 JMP 1604
1350 JMP 1604
1351 JMP 1557
1352 JMP 1570
1353 LDA 13
1354 APO
1355 JMP 1367
1356 LDA 12
1357 BCO↑
1360 -3777
1361 STA 12
1362 JMP 1616
1363 SRO
1364 1360
1365 JMP 1376
1366 JMP 1347
1367 AZE
1370 JMP 1363
1371 BCO 14
1372 AZE↑
1373 APO
1374 JMP 1363
1375 JMP 1356
1376 SRO↑
1377 2525
```

Multiply continued.

Divide the Fac by the Arg.
Leave the result in the Fac.

LISTING 2 (cont.)

```
1400 JMP 1346
1401 LDA 12
1402 STC 1125
1403 ADD 1131
1404 ROR 1
1405 JMP 1522
1406 JMP 1311
1407
1410 JMP 1557
1411 JMP 1744
1412 JMP 1557
1413 JMP 1046
1414 JMP 1200
1415 JMP 1046
1416 JMP 1616
1417 JMP 1412
1420 JMP 1253
1421
1422 JMP 1320
1423 360
1424 JMP 1730
1425 JMP 1320
1426 JMP 1060
1427 JMP 1414
1430 JMP 1060
1431 JMP 1420
1432 JMP 1060
1433 JMP 1422
1434 JMP 1060
1435 JMP 1424
1436 JMP 1557
1437 JMP 1076
1440 JMP 1060
1441 JMP 1412
1442 JMP 1625
1443 JMP 1546
1444 JMP 1700
1445 1776
1446 JMP 1706
1447 7776
1450 JMP 1711
1451 1
1452 JMP 1200
1453 JMP 1456
1454 JMP 1616
1455 JMP 1414
1456 JMP 1716
1457 JMP 1046
1460 JMP 1632
1461 JMP 1412
1462 JMP 1632
1463 JMP 1416
1464 JMP 1060
1465 JMP 1454
1466
1467
1470
1471
1472
1473
1474 LDA 15
1475 BCO 13
1476 SCR 13
1477 JMP 0
```

Divide continued.

0	Sqrt
1	Cla
2	Add
3	Com
4	Mul
5	Fac/Arg
6	Arg/Fac
7	Fac + I
10	Fac x I
11	Fac / I
12	I / Fac
13	Fix
14	Float
15	Clear
16	Max
17	Min
20	Sign
21	Increment
22	Subtract
23	Store
24	Set sign plus
25	Set sign minus
26	Fac - I

Jump Table

See page for the explanation of codes.

Temporary registers for square root.

Compare signs of Fac and Arg.

```
1500 LDA 15
1501 ROL 1
1502 BCO 15
1503 APO
1504 JMP 0
1505 SET 12
1506 0
1507 LDA 15
1510 AZE
1511 JMP 1515
1512 BCO 16
1513 APO↑
1514 AZE
1515 XSK↑12
1516 JMP 12
1517 CLR
1520 LDA 15
1521 SCR↑1
1522 STC 1124
1523 ADD 1125
1524 ROR↑1
1525 STC 1125
1526 ADD 1451
1527 ADD 1123
1530 STC 1123
1531 JMP 0
1532 LDA 16
1533 ROL↑1
1534 STA 16
1535 LDA 15
1536 ROL↑1
1537 STC 1124
1540 LAM 16
1541 CLR
1542 ADD 1447
1543 ADD 1123
1544 STC 1123
1545 JMP 0
1546 JMP 1557
1547 JMP 1456
1550 LDA 15
1551 STC 1127
1552 LDA 16
1553 STC 1130
1554 ADD 1123
1555 STC 1126
1556 JMP 0
1557 LDA 15
1560 STC 1121
1561 LDA 16
1562 STC 1122
1563 ADD 1123
1564 STC 1120
1565 JMP 0
1566 ADD 1126
1567 STC 1123
1570 ADD 1127
1571 STC 1124
1572 ADD 1130
1573 STC 1125
1574 JMP 0
1575 LDA 13
1576 STC 1127
1577 ADD 1122
```

If Arg is normalized return p+1.

If Arg equals zero return p+1,
otherwise return p+2.

Scale Arg right, increment exponent.

Scale Arg left, decrement exponent.

Put Arg into Fac and return to store.

Arg → B

Arg → Fac

B → Arg

LISTING 2 (cont.)

```
1600 STC 1130
1601 ADD 1120
1602 STC 1126
1603 JMP 0
1604 CLR
1605 LDA 14
1606 LAM 16
1607 LDA 13
1610 LAM 15
1611 STC 1045
1612 LAM 16
1613 STC 1045
1614 LAM 15
1615 JMP 0
1616 LDA 16
1617 COM
1620 STC 1125
1621 ADD 1124
1622 COM
1623 STC 1124
1624 JMP 0
1625 CLR
1626 STC 1123
1627 STC 1124
1630 STC 1125
1631 JMP 0
1632 SET 12
1633 0
1634 LDA 13
1635 APO↑
1636 JMP 1644
1637 COM
1640 STA 13
1641 LDA 14
1642 COM
1643 STA 14
1644 LDA 15
1645 APO
1646 JMP 1616
1647 JMP 12
1650 LDA
1651 0
1652 STC 1672
1653 LDA 15
1654 SCR 13
1655 STA
1656 1664
1657 LAM 16
1660 LDA 15
1661 SCR 13
1662 LAM 15
1663 BCO↑
1664 0
1665 APO↑
1666 JMP 1671
1667 ROR↑1
1670 JMP 1522
1671 CLR
1672
1673 SRO↑
1674 0
1675 JMP 1616
1676 JMP 1557
1677 JMP 1046
```

Fac → B

Arg + Fac → Arg

Complement Arg

Clear Arg

Complement Arg, or Fac if they are negative.

Round off Arg.

Restore sign following multiply and divide.

LISTING 2 (cont.)

```
1700 JMP 1133
1701 JMP 1046
1702 JMP 1575
1703 JMP 1557
1704 JMP 1566
1705 JMP 1046
1706 JMP 1133
1707 JMP 1702
1710 JMP 1046
1711 JMP 1505
1712 JMP 1715
1713 LDA 15
1714 SCR 12
1715 JMP 1046
1716 SET 12
1717 1026
1720 LDA
1721 1120
1722 STA 12
1723 LDA 13
1724 STA↑12
1725 LDA 14
1726 STA↑12
1727 JMP 0
1730 LDA
1731 0
1732 STC 1736
1733 JMP 1575
1734 JMP 1557
1735 JMP 1566
1736
1737 LDA 13
1740 SCR↑14
1741 BCO↑
1742 3777
1743 JMP 1046
1744 ADD 17
1745 STC 1407
1746 JMP 1000
1747 ~~SAE↑11~~ 1471
1750 23
1751 CLR
1752 ADD 1120
1753 SCR 1
1754 STC 1120
1755 JMP 1000
1756 1466
1757 4023
1760 1471
1761 4006
1762 1466
1763 4002
1764 1777
1765 11
1766 SRO↑
1767 3567
1770 JMP 1755
1771 ADD 1407
1772 STC 17
1773 LDA 17
1774 APO
1775 JMP 1014
1776 JMP 1051
1777 2
```

If Arg ≥ Fac, put it into the Fac.

If Arg ≤ Fac, put it into the Fac.

Check sign of arg.

Store Fac.

Put Arg → Fac and Fac → Arg.
ie: reverse the two quantities.

Set accumulator to plus or minus 3777.

Square root routine

IV. SUBROUTINES SUITABLE FOR QUESTION-ANSWER PROGRAMMING OF THE LINC COMPUTER

This section contains a number of subroutines useful to the LINC programmer in constructing question-answer programs providing for computer-user interaction. The subroutines are in symbolic coding form, suitable for inclusion with the programmer's master program, and intended to be assembled into LINC code at the time of complete program assembly.

The subroutines were written in the LASS assembly language for use with the LOSS operating system on the LINC (see Section V). They take care of the timing and data conversion necessary to input or output alphanumerics, octal numbers, or decimal numbers from or to a teletype. The teletype used in development of these subroutines is a Model 33 (unbuffered) connected to the LINC external level 0 for input and relay 0 for output to the teletype.

Generally, six functions are implemented:

(1) Alphanumeric character table output to teletype.

(2) Teletype alphanumerics into a character table.

(3) Octal number in the LINC accumulator output to teletype and typed in octal.

(4) Teletype input of an octal number to an octal number in the LINC accumulator.

(5) Octal number in the LINC accumulator output to teletype and typed in decimal (12 bit or 24 bit).

(6) Teletype input of decimal integers converted to octal in the LINC accumulator (12 bit or 24 bit).

Certain other functions such as RUBOUT, line feed, carriage return, etc., are included. These are all detailed in the individual subroutine descriptions in this report.

Each subroutine description includes:

(1) Specification of the index registers, if any, the subroutine uses. (The index registers used may be used elsewhere, but the contents of these registers is subject to destruction when the program enters this subroutine.)

(2) The entrance tag. (It must be declared "GLOBAL" in the calling program in the LOSS system.)

(3) Other subroutines of the set which may be called by the described subroutine.

(4) The number of locations used by the described subroutine.

(5) A typeout of the LASS symbolic coding.

Since LAP4 has certain format and assembly limitations, several changes must be made to the LASS symbolic coding. The "GLOBAL" tag declaration has no function in LAP4 and must be removed when the coding is inserted into the LAP4 manuscript. The "i bit" designation ; must be changed to i, and the "present line" indicator . must be changed to p. Since the LASS assembler permits tags of any length the LASS tags must be converted to the two character "number,letter" format and preceded by a "#" rather than a comma when declared.

The descriptions that follow refer to the location of the jump instruction to the subroutine as "call" and the following locations are "call + 1," "call + 2"....

All numbers appearing in the descriptions are octal unless otherwise indicated.

1. Subroutine: READCHAR

Index Reg. Used: 0, 17

Number of Locations: 43

Entrance Tag: RC

Other Subroutined Called: None

READCHAR accepts one teletype character and puts the appropriate code in the accumulator. Entering the READCHAR subroutine at tag RC initiates a loop waiting for a key to be struck on the teletype. Upon exit, the half word code (see Table 3 at the end of this section) representing the character will be in the right half of the accumulator. Control is returned to the main program two instructions beyond the jump to RC (call + 2).

Three exceptions exist to the above rules. Striking the "RUBOUT" key causes a 137 to be left in the accumulator when control is returned to call + 2. A carriage return or line feed will cause control to be returned to main program at the instruction following the jump to RC (call + 1) though nothing of any significance will be left in the accumulator.

No provision has been made for using the special control keys.

The symbolic coding for READCHAR is given in Listing 3.

```
**READCHAR**

GLOBAL RC
.RC LDA  (ENTER HERE)
0
STC TERM  (HALF)
XSK;0 (WORD CODE)
ADD 0    (IS)
STC RETN (RETURNED)

<10>
SXL 0   (IN ACCUM)
JMP.-1
CLR (RETURNS TO CALL+1)
SET;17 (ON CAR. RET.)
1241 (OTHERWISE CALL+2)
.PULSE BSE;
200  (I17 USED TEMPOR)
.WAIT XSK;17

<20>
JMP.-1 (PULSES COME)
LZE  (IN ON EXTERNAL)
JMP GOT (LEVEL 0)
ROR;1  (FROM UNBUFF)
SET;17  (TELETYPE)
1340
SXL 0
JMP WAIT

<30>
JMP PULSE
.GOT XSK;17 (EXTRA)
JMP.-1      (PAUSE)
ROL;1
COM
ADA;
277
AZE;

<40>
CLR
APO
.TERM  (EXITS HERE)
SCR 1 (ON CR OR LF)
.RETN (NORMAL EXIT)
```

LISTING 3

2. Subroutine: DECOCT

Index Reg. Used: 0

Number of Locations: 55

Entrance Tag: DO

Other Subroutines Called: READCHAR

DECOCT accepts a string of teletype numerics, interprets them as a decimal number, and puts the 12 bit octal conversion in the accumulator. Entering DECOCT at tag DO initiates a loop waiting for a decimal integer to be input from the teletype. Spaces are ignored. A minus sign is necessary for negative quantities but the plus sign is optional. The entry is considered terminated when a carriage return is struck. At this time control is transferred to the main program at "call + 2" with the quantity in octal left in the accumulator.

When a "RUBOUT" or other non-numeric (other an +- or space) character is struck, control is returned to the main program at "call + 1."

No check is made for overflow of the 12 bit LINC word.

The symbolic coding for DECOCT is given in Listing 4.

```
**DECOCT**

GLOBAL DO RC
,DO LDA  (CONVERTS)
0  (A TYPED DECIMAL)
STC ERROR (QUANTITY)
XSK;0 (TO A 12 BIT)
ADD 0 (OCTAL NUMBER)
STC RETN (IN ACCUM)

<10>
,MINUS STC FLAG
,NEXT STC ANS
JMP RC (IGNORES SPACES)
JMP DONE
AZE;
JMP.-3
SHD; (CHECK FOR SIGNS)
1300

<20>
JMP.-6
SHD;
1500
JMP MINUS
SAE;  (NORMALLY)
137 (RETURNS TO)
JMP.+2 (CALL+2)
JMP ERROR (BUT ON)

<30>
ADA; (INPUT FORMAT)
-31  (ERROR IT EXITS)
APO;  (TO CALL+1)
,ERROR 000
ADA;
11
AZE
APO;   (USE WITH)

<40>
JMP.+2 (READCHAR)
JMP ERROR (SUBROUTINE)
STC TEMP
ADD ANS
MUL;  (NO CHECK FOR)
12     (OVERFLOW)
ADA;
,TEMP

<50>
JMP NEXT (JUMPS)
,DONE LDA; (<HERE>)
,ANS   (ON CAR RET)
SRO;
,FLAG  (MINUS FLAG)
COM
,RETN
```

LISTING 4

3. Subroutine: DBLDECOCT

Index Reg. Used: 15, 16

Number of Locations: 140

Entrance Tag: DDO

Other Subroutines Called: READCHAR

DBLDECOCT accepts a string of teletype numerics, interprets them as a decimal number, and puts the 24 bit octal conversion in locations "call + 1" (most significant) and "call + 2" (least significant). Spaces are ignored. A minus sign is necessary for negative quantities but the plus sign is optional. The entry is considered terminated when a carriage return is struck. At this time control is transferred to the main program at "call + 4" with the quantity in "call + 1 and call + 2."

When a "RUBOUT" or other non-numeric (other than + - or space) characters is struck, control is returned to the main program at "call + 3."

No check is made for overflow of the 12 bit LINC word.

The symbolic coding for DBLDECOCT is given in Listing 5.

```
**DBLDECOCT**

GLOBAL DDO RC
,DDO LDA (ENTER HERE)
0  (ACCEPTS A TYPED)
BCL; (DEC QTY)
6000 (RETURNS HIGH)
STC 15 (ORDER 12 BITS)
XSK;0 (TO CALL+1)

<10>
XSK;0(LOW TO CALL+2)
ADD 0
STC ERROR
XSK;0
ADD 0
STC RETN(I15 I16 USED)
,MINUS STC FLAG
STC ANSL

<20>
STC ANSU
,NEXT STC TEMP1
STC TEMP2
JMP RC (SKIPS SPACES)
JMP DONE
AZE;
JMP.-3
SHD; (CHECK FOR SIGNS)

<30>
1300
JMP.-6
SHD;
1500
JMP MINUS
SAE;  (NORMALLY)
137 (RETURNS TO)
JMP.+2 (CALL+4)

<40>
JMP ERROR (BUT ON)
ADA; (INPUT FORMAT)
-31  (ERROR IT EXITS)
APO; (CALL+3)
,ERROR 000
ADA;
11
AZE

<50>
APO;   (USE WITH)
JMP.+2 (READCHAR)
JMP ERROR (SUBROUTINE)
AZE;
CLR
STC TEMP
SET;16
-12

<60>
,MULT12 CLR
ADD ANSL
LAM;
,TEMP1
LDA;
,ANSU
LAM;
,TEMP2

<70>
STC 0
LAM
TEMP1
STC 0
LAM
TEMP2
XSK;16
JMP MULT12

<100>
CLR
ADD TEMP
LAM
TEMP1
STC 0
LAM
TEMP2
STC 0

<110>
LAM
TEMP1
STC ANSL
LAM
TEMP2
STC ANSU
JMP NEXT (JUMPS)
,DONE SRO;
```

LISTING 5

```
<120>
,FLAG
JMP COMP
LDA
ANSU
STA 15
LDA
ANSL
STA 15

<130>
,RETN
,COMP LDA
,ANSL
COM
STC ANSL
ADD ANSU
COM
STC ANSU

<140>
JMP FLAG+2
,TEMP
```


LISTING 5 (Cont.)

4. Subroutine: OCTALIN

Index Reg. Used: 0

Number of Locations: 53

Entrance Tag: OCTIN

Other Subroutines Called: READCHAR

OCTALIN accepts a string of teletype numerics, interprets them as an octal number, and puts the octal number in the accumulator. Entering OCTALIN at tag OCTIN initiates a loop waiting for an octal integer to be input from the teletype.

Spaces are ignored. A minus sign causes the number entered to be complemented (typing - 136 will give the same result as typing 7641). A plus sign is optional. The entry will be considered terminated when a carriage return is struck. At this time control is returned to the main program at "call + 2" with the quantity left in the accumulator.

When a "RUBOUT" or any character other than +, -, 0, 1, 2, 3, 4, 5, 6, 7 is struck control is returned to the main program at "call + 1."

No check is made for overflow of the 12 bit LINC word.

The symbolic coding for OCTALIN is shown in Listing 6.

```
**OCTALIN**

GLOBAL OCTIN RC
,OCTIN LDA (ACCEPTS)
0  (DIGITS TYPED IN)
STC ERROR   (AND)
XSK;0   (FORMS AN OCTAL)
ADD 0
STC RETN (WORD IN)

<10>
,MINUS STC FLAG (ACCUM)
,NEXT STC ANS
JMP RC
JMP FIN
AZE; (IGNORE SPACES)
JMP.-3
SHD;
1400 (CHECK FOR SIGN)

<20>
JMP.-6
SHD;  (NORMALLY RETURNS)
1500   (TO CALL+2)
JMP MINUS (BUT JUMPS)
SAE;  (TO CALL+1 ON)
137  (RUBOUT AND )
JMP.+2 (IMPROPER)
JMP ERROR (INPUT)

<30>
ADA;     (FORMAT)
-27
APO;
,ERROR 0000
ADA;
7  (USE WITH READCHAR)
AZE   (SUBROUTINE)
APO;

<40>
JMP.+2
JMP ERROR (NO CHECK)
ADA;   (FOR OVERFLOW)
,ANS 0000
ROL 3
JMP NEXT  (JUMPS)
,FIN LDA (<HERE>)
ANS   (ON CAR. RET.)

<50>
ROR 3
SRO;
,FLAG   (MINUS FLAG)
COM
,RETN
```

LISTING 6

5. Subroutine: ALNUMIN

Index Reg. Used: 0, 15, 16

Number of Locations: 34

Entrance Tag: ANIN

Other Subroutines Called: READCHAR

ALNUMIN accepts a string of teletype alphanumerics and places the appropriate codes in a defined character table in LINC core.

Entering ALNUMIN at tag ANIN initiates a loop which waits for alphanumeric characters to be typed in and stores these characters in half word code (see Table 3 at the end of this section). These codes are stored (two per word) in core as designated by index register 15. Index register 15 should be set to the address of the first word of the storage area (called TABLE in sample below) + 3777. The first character typed goes into the left half of the first word of the storage area; the second character goes into the right half, etc. Index register 16 should be set to the negative of the maximum number of characters intended for input. Leading spaces are suppressed during input and need not be counted.

The routine returns to the main program at "call + 3" when a carriage return is struck to "call + 2" when the "RUBOUT" key is struck, and "call + 1" when the number of characters entered exceeds the size of the table as indicated by index register 16.

The following sample call provides for input of 12_8 characters:

```
SET:  15                                  ,TABLE 0
TABLE + 3777                                0
SET:  16                                    0
-12                                         0
JMP ANIN                                    0
JMP OVERFLOW (too many characters)
JMP CANCEL ("RUBOUT" key struck)
    °
```

```
**ALNUMIN**

GLOBAL ANIN RC
,ANIN LDA (ENTER HERE)
0  (SET I16 TO -MAX NO)
STC MAX (OF CHARACTERS)
XSK;0
ADD 0
STC RUBOUT

<10>
XSK;0 (SET I15 TO)
ADD 0 (YOUR TABLE+3777)
STC RETN(THE TABLE WILL)
JMP RC (BE FILLED TWO)
JMP RETN  (CHARS/WORD)
SAE;
137
JMP.+2 (USE WITH)

<20>
,RUBOUT 00 (READCHAR)
AZE;   (SUBROUTINE)
JMP.-7
JMP ST
,NEXT JMP RC
,RETN (NORMALLY)
SAE;  (RETURNS T0)
137 (CALL+3 BUT ON)

<30>
JMP.+2 (TABLE OVERFLOW)
JMP RUBOUT (EXIT IS TO)
,ST STH;15 (CALL+1)
XSK;16 (AND ON RUBOUT)
JMP NEXT (IT RETURNS)
,MAX (TO CALL+2)
```


LISTING 7

6. Subroutine: TYPECHAR

Index Reg. Used: 0, 12, 13, 14
Number of Locations: 30
Entrance Tag: TC
Other Subroutines Called: None

Entering TYPECHAR at tag TC causes the typing of the character whose half word code (see Table 3) is in the accumulator. If one wishes to perform a carriage return or line feed, the actual teletype code (see Table 4 at the end of this section) must be put in the accumulator and the subroutine entered at tag CRLF.

In either of the above cases the subroutine returns to the main program at the instruction following the jump to entrance.

The symbolic coding for TYPECHAR is given in Listing 8.

```
**TYPECHOR**

GLOBAL TC CRLF
.TC ROL 1 (ENTER HERE)
COM  (WITH HALF WORD)
ADA;  (CODE IN ACCUM)
277
.CRLF SET 12 (CAR RET)
0  (AND LINE FEED)

<10>
SET;13 (SPECIAL)
-13 (JMP CRLF WITH)
STC TEMP (TTY CODE)
.NEXT LDA; (IN ACCUM)
.TEMP 0
ROR;1
STC TEMP
ROL;1  (OUTPUT)

<20>
ATR (THRU RELAY 0)
SET;14
1356  (I12 I13 I14)
XSK;14 (USED TEMPOR)
JMP.-1
XSK;13
JMP NEXT
XSK;13  (EXTRA)

<30>
JMP.-1  (WAIT)
JMP 12 (RETURN)
```

LISTING 8

7. Subroutine: OCTDEC

Index Reg. Used: 0, 12, 13
Number of Locations: 73
Entrance Tag: OD
Other Subroutines Called: TYPECHAR

Entering OCTDEC at tag OD with a quantity in the accumulator will cause that quantity to be converted to a decimal integer and typed out. Negative quantities will be preceded by a minus sign. The number will be right justified in a four place field with zero suppression carried up to but not including the last place.

The subroutine returns to the main program at the instruction following the jump to the entrance.

The symbolic coding for OCTDEC is given in Listing 9.

```
**OCTDEC**

GLOBAL OD TC
,OD STC HOLD (ENTER)
ADD 0 (HERE WITH QTY)
STC RETN (IN ACCUM)
STC DIGITS
STC DIGITS+1
STC DIGITS+2

<10>
STC DIGITS+3
STC DIGITS+4
STC FLAG
ADD HOLD (USE WITH)
AZE;(TYPECHAR SUBRTN)
CLR (I12 I13 USED TEMP)
APO;
JMP POS (IF NEG COM)

<20>
COM    (AND)
STC HOLD
ADD.-2 (SET - FLAG)
STC FLAG
,POS SET;12
DIGITS-1
LDA;
,HOLD

<30>
SET;13
1777
,CONV XSK;13
ADA;
-12
APO
AZE;
JMP CONV

<40>
ADA;
32   (12+20)
STA;12
LDA
13
AZE
JMP CONV-2
LDA;

<50>
15
SRO; (CHECK)
,FLAG
STA;12 (-FLAG)
CLR
ADD DIGITS+4
JMP TC
ADD DIGITS+3

<60>
JMP TC
ADD DIGITS+2
JMP TC
ADD DIGITS+1
JMP TC
ADD DIGITS
JMP TC
,RETN

<70>
,DIGITS  (THE SUBRTN)
0    (PUTS THE DIGITS)
0    (HERE IN HALF)
0   (WORD CODE)
0
```


LISTING 9

8. Subroutine: DBLOCTDEC

Index Reg. Used: 0, 15, 16

Number of Locations: 202

Entrance Tag: DOD

Other Subroutines Called: TYPECHAR

Entering OCTDEC at tag DOD with a 24 bit quantity held in locations "call + 1" (most significant) and "call + 2" (least significant) will cause that quantity to be converted to a decimal integer and typed out. Negative quantities will be preceded by a minus sign. The number will be right justified in an eight place field with zero suppression carried up to but not including the last place.

The subroutine returns to the main program at location "call + 3."

The symbolic coding for DBLOCTDEC is given in Listing 10.

```
**DBLOCTDEC**

GLOBAL DOD TC
.DOD LDA (ENTER HERE)
0 (WITH HIGH ORDER)
BCL; (12 BITS IN CALL+1)
6000 (LOW ORDER IN)
STC 15 (CALL+2)
LDA 15 (RETURNS TO)

<10>
STC QTYU (CALL+3)
LDA; 15(AFTER TYPING)
STC QTYL (7 DIGIT DEC)
XSK; 0 (NUMBER)
XSK; 0
ADD 0 (I15,16 USED)
STC RETN
STC FLAG

<20>
SET; 16
DIGITS-1
SET; 15
-10
STA; 16
XSK; 15
JMP.-2
ADD QTYU (USE WITH)

<30>
AZE (TYPECHAR SUBRTN)
JMP CHKPOS
ADD QTYL
AZE
JMP CHKPOS
CLR
STC QTYU (KILL -0'S)
STC QTYL

<40>
JMP POS
.CHKPOS LDA
QTYU
APO; (IS VALUE POS?)
JMP POS (YES)
COM (NO)
STC QTYU
ADD QTYL

<50>
COM
STC QTYL(SEND A)
LDA; (- FLAG TO)
1 (LOC FLAG BELOW)
STC FLAG
.POS SET; 16
DIGITS-1
.RESET LDA; (SET)

<60>
7777(DIVIDE QUOTIENT)
STC CTRU (COUNTER)
LDA;
7776
STC CTRL
.CONV CLR
LDA; (INCRE)
1  (DIVIDE COUNTER)

<70>
LAM;
.CTRL
STC 0
LAM;
.CTRU
STC 0
LAM
CTRL

<100>
STC 0
LAM
CTRU
CLR
LDA; (DIVIDE BY 12)
-12
LAM;
.QTYL

<110>
LDA;
7777
LAM;
.QTYU
STC 0
LAM
QTYL
STC 0
```

LISTING 10

```
<120>
LAM
QTYU
APO;
JMP CONV
AZE
JMP.+4
ADD QTYL
AZE;

<130>
JMP CONV
CLR (CONVERT THE)
ADD QTYL (REMAINDER)
ADA; (TO OCTAL CODE)
32
STA;16 (STORE DIGIT)
LDA (DONE?)
CTRL

<140>
STC QTYL
ADD CTRU
STA
QTYU
AZE
JMP RESET (NO)
LDA
CTRL

<150>
AZE
JMP RESET (NO)
LDA;
15
SRO; (CHECK FOR)
,FLAG (- FLAG)
STA;16
SET;15 (YES)

<160>
-10
SET;16
DIGITS+7
LDA 16 (SEND DIGITS)
JMP TC (TO TYPECHAR)
LDA;
-1
ADM

<170>
16
XSK;15
JMP.-7
,RETN
,DIGITS 0
DITTO 7
```

LISTING 10 (Cont.)

9. Subroutine: OCTALOUT

Index Reg. Used: 0, 15

Number of Locations: 22

Entrance Tag: OCTOUT

Other Subroutines Called: TYPECHAR

Entering OCTALOUT at the tag OCTOUT with a quantity in the accumulator will cause that quantity to be typed out as a positive octal integer in the range [0000...7777].

The subroutine returns to the main program at the instruction following the jump to the entrance.

The symbolic coding for OCTALOUT is given in Listing 11.

```
**OCTALOUT**

GLOBAL OCTOUT TC
,OCTOUT STC HOLD(ENTER)
ADD 0    (HERE WITH)
STC RETN (OCTAL QTY)
SET;15 (IN ACCUM)
7773
,ROTATE LDA;

<10>
,HOLD (I15 USED TEMPOR)
ROL 3
STA
HOLD
BCL;  (USE WITH)
7770 (TYPECHAR)
ADA;  (SUBROUTINE)
20

<20>
JMP TC
XSK;15
JMP ROTATE
,RETN
```

LISTING 11

10. Subroutine: ALNUMOUT

Index Reg. Used: 0, 15, 16
Number of Locations: 10
Entrance Tag: ANOUT
Other Subroutines Called: TYPECHAR

Entering ALNUMOUT at tag ANOUT will cause a set of alphanumeric characters as determined by index registers 15 and 16 to be typed out. Index register 15 should be set to the first word + 3777 of the set of words containing the half word codes of the characters to be typed. Index register 16 should be set to the negative of the total number of characters to be typed.

The subroutine returns to the main program at the location following the jump to the entrance.

The symbolic coding for ALNUMOUT is given in Listing 12.

The following sample call provides for typing the clause THIS IS A TEST.

```
SET; 15
TABLE + 3777
SET; 16
-16
JMP ANOUT
   •
   •
   •
,TABLE 6450 (TH)
5163        (IS)
0051        ( I)
6300        ( S)
4100        ( A)
6445        (TE)
6364        (ST)
```

```
**ALNUMOUT**

GLOBAL ANOUT TC
,ANOUT LDA (SET I15 TO)
0 (YOUR TABLE+3777)
STC RETN (SET I16 TO)
LDH;15 (-NO. OF CHARS)
JMP TC (PUT 2 CHAR/WD)
XSK;16 (IN THE TABLE)

<10>
JMP.-3 (IN 1/2 WD FORM)
,RETN (USE W/TYPECHAR)
```

LISTING 12

11. Subroutine: ENDLINE

Index Reg. Used: 0

Number of Locations: 12

Entrance Tag: EOL

Other Subroutines Called: TYPECHAR

Entering ENDLINE at tag EOL causes the teletype to execute a carriage return and a line feed.

The subroutine returns to the main program at the location following the jump to the entrance.

The symbolic coding for ENDLINE is given in Listing 13.

```
**ENDLINE**

GLOBAL EOL CRLF
,EOL LDA (ENTER HERE)
0    (WILL CAUSE A)
STC RETN (CARR. RET.)
LDA;  (AND LINE FEED)
345
JMP CRLF

<10>
LDA;  (USE WITH)
353   (THE TYPECHAR)
JMP CRLF (SUBROUTINE)
,RETN
```

LISTING 13

TABLE 3

CHARACTER	CODE	CHARACTER	CODE
Blank	00	@	40
!	01	A	41
"	02	B	42
#	03	C	43
$	04	D	44
%	05	E	45
&	06	F	46
'	07	G	47
(	10	H	50
)	11	I	51
*	12	J	52
+	13	K	53
,	14	L	54
-	15	M	55
.	16	N	56
/	17	O	57
0	20	P	60
1	21	Q	61
2	22	R	62
3	23	S	63
4	24	T	64
5	25	U	65
6	26	V	66
7	27	W	67
8	30	X	70
9	31	Y	71
:	32	Z	72
;	33	Carr.Ret.	73
<	34		74
=	35		75
>	36	↑	76
?	37	←	77

Derived by:

```
CLR
ADD teletype-code
COM
ADA;
277
SCR1
```

TABLE 4

TELETYPE CODES

A	175	?	201	1	235
B	173	←	101	2	233
C	171	>	203	3	231
D	167	=	205	4	227
E	165	↑	103	5	225
F	163	<	207	6	223
G	161	;	211	7	221
H	157	]	105	8	217
I	155	:	213	9	215
J	153		107	Ø	237
K	151	[	111	SPACE	277
L	147	.	243	RETURN	345
M	145	-	245	ADVANCE	353
N	143	,	247		
O	141	+	251		
P	137	*	253		
Q	135	)	255		
R	133	(	257		
S	131	'	261		
T	127	&	263		
U	125	/	241		
V	123	%	265		
W	121	$	267		
X	117	#	271		
Y	115	"	273		
Z	113	!	275		

V. THE LOSS SYSTEM

This section presents a general description of the LOSS system described in detail in NASA Technical Report No. IRL-1038 (see bibliography). It includes basic information about certain LOSS functions, the symbolic assembler (LASS), and the handling of non-LOSS structured data tapes under LOSS.

The LINC Operating System, LOSS, consists of a basic control program (the monitor) and additional programs to provide for writing and running programs without dealing directly with the controls of the computer each time a program is run. This type of system is generally referred to as an executive system, a monitor system, an operating system or some combination of these terms. In the case of LOSS the above functions are carried out using tape unit 0 (left hand tape unit) for storage of all absolute programs (the program stack) and reserving tape unit 1 for data storage.

1. The LOSS Program Stack and Its Index

The LOSS program stack is made up of the basic system's programs together with the programs assembled onto it by the individual user. This set of programs is located on the micro tape which, under the LOSS system, is always mounted on tape unit 0 (left hand tape unit). Each program normally occupies one of 77 (all numbers are in octal) stack positions on this tape. A stack position is a 10 block area on tape. Stack position 1 consists of blocks 10-17; position 2 consists of blocks 20-27 . . . position 77 consists of blocks 770-777.

The system's programs occupy stack positions 10 through 25 (blocks 100-257). Though there are 16 stack positions for the system's programs only 10 programs actually exist (the remaining stack positions are used as working storage areas and additional program stack locations needed by some of the system's programs). Several of these system's programs merit special consideration; MONITOR, DEFINE, EDIT and LASS.

If the user simply intends to run previously written programs, then only an understanding of the Monitor and Define programs are necessary. An understanding of the Edit and Lass programs enable the user to write and operate his own programs.

The basic set of system's programs on the program stack also includes several which could be considered utility programs; TYPE, DISPLAY, and DISTAPE. These are described on pages 16-24 of the LOSS manual.

2. Monitor (MNTR)

Though the monitor exists on the program stack (position 15) in the same manner as all other programs, it is unique in that it forms the basis for the whole LOSS system.

It is loaded by setting the left and right switches to 0700 0150 (RDC 150) then lifting the DOTOG lever -- the right switches are then set to 0000 and the START RS button is pushed.

The functions of the Monitor are: (1) to accept a program name (or number -- see below) and relate it to a program stack position; (2) to accept numeric (in octal) and alphanumeric (see string format) parameters and relay them to the program being called (methods for accepting these parameters by the user's program are discussed in the notes on the assembler); (3) to read in additional tape blocks 1, 2, 3, 4, 5 of the stack position into quarters 1-5 of the memory and jump to location 400 to begin execution. (Quarters 0 and 6 are intended to be left undisturbed. However, steps can be taken to use those quarters also -- see LASS below.); (4) upon completion of a program (a JMP RETURN in the program, RETURN is recognized by the assembler as a reserved word related to a routine in quarter 0), execution returns to the undisturbed quarter 0 which reads in quarters 1-5 of the monitor which in turn requests another program call.

Other more obscure tasks, OVERLAY, PARAMS, SAVE-RESTORE are performed by the quarter 0 portion of the monitor and are discussed on

pages 39 to 43 of the LOSS manual.

The index to the program stack resides on block 155 of the system's tape and comes into core along with the rest of the monitor. Its structure is such that the monitor can easily correspond name-parameter entries thereon to absolute program stack positions (further explanation under Updating the Program Stack Index). In its basic form the program stack index contains only the names of the system's programs. The user may update this index using the procedure discussed later (it is not done automatically when assembling a program into a stack location). Since programs can be called by stack location number (a parameter given at assembly time), it is not absolutely necessary to update the index.

3. DEFINE and the Data Tape (Unit 1)

Before discussing the program DEFINE, used in the allocation of data storage areas on unit 1, a thorough explanation of the LOSS data storage system must be made. The micro tapes upon being marked (see Lap 4 manual) are divided into 1000 (octal) blocks of 400 (octal) words each. The LOSS system considers the 1000 blocks as 10 groups of 100 blocks each. Each group of 100 blocks is called a book. The first block of each book; 0, 100, 200 . . . 700 is reserved as an index for that particular book. As areas of a book are allocated using the DEFINE program (see below) entries are automatically made in its index consisting of a name, a starting block number, and a length. These allocated areas are generally referred to as Texts or Files and apply only to tape unit 1. Examples of their use are shown later.

The DEFINE program requires from the user (1) the number of the book in which he wishes to reserve storage space, (2) the name the user wishes to associate with the file, (3) the number of blocks of space he desires to reserve (of course, must be less than 77 blocks). These three parameters must be entered in the form of an alphanumeric string (see page 4 of LOSS manual for general definition of a string). The following is an example of the use of the DEFINE program.

<PROG>>	Typed by monitor.
DEFINE↵	Program name typed by user (↵ indicates carriage return).
	(__ Underline indicates use of reserved word or symbol.)
1 STRING>>	Parameter request typed by monitor.

%"BOOK 4 DEFINE TESTDATA 16"↵

;↵	Semicolon and carriage return by user indicates the end of parameters.
<RUN>>	
<PROG>>	Indicates successful completion by calling for next program.

The above example would reserve a file (on the tape on unit 1) in book 4, with a length of 16 blocks which could be referred to in some other program call by:

/4↵	Slash followed by book number.
TESTDATA↵	

The monitor system would convert the name and book number (using the information in the book index) to an absolute starting block number (somewhere between 401 and 477) and a length (16 in this case) and make these two octal parameters available to a user's program. Reference the section Accepting Parameters for programming methods for accepting these parameters into a user program.

The DEFINE program also provides for relinquishing unneeded text (files) space on unit 1. The input is again a string of the following type:

<PROG>>	Program request.
DEFINE↵	Program name typed.
1 STRING>>	Parameter request.

```
%"BOOK 4 ERASE TESTDATA"     String entry made by user.
;                            Terminator typed.
<RUN>>
<PROG>>                      Next program requested.
```

The previous example would cause the DEFINE program to go to the book 4 index (block 400) and remove the entry which is reserving a particular set of blocks in that book under the name TESTDATA. It should be remembered that in neither the define nor erase mode is any action taken on the file (text) itself -- only on the particular book index for that area.

Several defines and/or erasures can be combined as follows when they refer to the same book.

```
<PROG>>
DEFINE
1 STRING>>
%"BOOK 4
ERASE OLDDATA
DEFINE DATASPACE
DEFINE OUTDATA"
;
<RUN>>
<PROG>>
```

4. EDIT

The EDIT program is generally used to write symbolic programs which later will be converted to absolute (machine language) programs by the assembler (see LASS below). The EDIT program is used both for

the initial writing of the symbolic program as well as later correction and updating of the program.

Edit requires that a file be defined to store the symbolic coding entered through the teletypewriter. This file may be a maximum of four blocks long. The number of instruction lines held in four blocks depends on the number of characters per line but generally is in the range of 400 to 500. However, more than one four block text can be used to hold the symbolic coding for a program (see CONT page 31 of the LOSS manual).

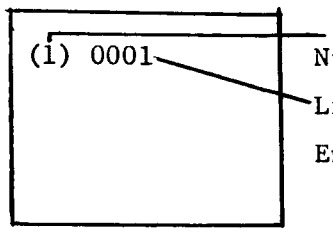
The EDIT program could also be used to develop alphanumeric text for other applications since it is simply a program which provides for storing BCD characters on tape (see page 9 of the LOSS manual for the LOSS BCD codes).

One such application is the updating of the system's tape program stack index (in no way related to book indexes on tape unit 1) which is described later on.

The EDIT program is called in the following manner:

<PROG>>	Monitor's program request.
EDIT↵	User asks for EDIT programs. Monitor
FILE>>	asks for file on which to store the
3↵	alphanumeric information.
SYMBPROG↵	User previously declared some file
;	(max 4 blocks) on book 3 in this example.
↵	

Presuming the user is just starting to write a symbolic program (not changing a previously written one) the EDIT program simply displays two numbers and the end of text symbol on the LINC oscilloscope (since no test material, coding, exists).



There are two basic modes of the EDIT program -- the Control mode (initial state) and the Input mode. The control mode is used to move through the text with the use of the oscilloscope and to perform certain operations on the text. The input mode is entered by striking an I on the teletypewriter while in the control mode. It is in the input mode that the actual alphanumeric (symbolic coding) material is entered. One exits from the input mode by striking a ! on the teletypewriter. A detailed description of the EDIT program and its control operations begins on page 12 of the LOSS manual.

5. LASS*

LASS is the symbolic assembler under the LOSS system. Though called just like any other program on the program stack, its function is to convert a symbolic program written with EDIT into an absolute (machine language) program and store that absolute program in a numbered program stack position as specified by the user. This new program can then be called through the monitor (by number only at this point) in the normal manner.

The LASS assembler is called in the following manner:

<PROG>>	Monitor's program request.
LASS	User types in program name.
STACK #, FILES	Monitor's parameter request.

* The actual language syntax is described on pages 26-34 with a list of operation mneumonics on page 38 of the LOSS manual. LINC, Vol. 16, Programming and Use-1 describes the basics of LINC programming.

32↵	User types in the stack location he
13↵	desires, the book number, and
SYMBPROG↵	file name of the symbolic coding developed with Edit.
;↵	

The stack location is the number of the program stack location on tape unit 0 on which the user wishes to have the absolute coding stored. Any stack location 0-77 will be accepted. However, the monitor nor LASS makes no provision for safeguarding a program previously assembled onto the stack location in question. In particular the user must realize that stack positions 10-25 are used by the system's programs themselves.

One of this set, position 22, is a working storage position for the LASS assembler and may be given as the stack location for an assembly. Such a program will stay intact on stack location 22 until another assembly is performed.

Performing an assembly in no way affects the program stack index. Until the stack number assembled into has been given a name (see Updating Program Stack Index), the program assembled can be referred to only by number.

For the previous example the call to execute the newly created program would be:

<PROG>>	Monitor's program request.
32↵	Number of program (instead of a name).
PARAMS>>	Parameter request given for programs called by number.

Parameters would be given as provided for in the particular program.

i

<RUN>>

<PROG

A. Program Starting Locations

The assembler, under normal circumstances, assembles the first symbolic code line such that it will be located in location 400 when the program is loaded by the monitor. All succeeding lines will be put in successively higher locations. This is equivalent to storing the absolute coding for the first 400 symbolic instructions on block 1 of the 10 block program stack position selected by the user. When the monitor loads the program it loads blocks 1, 2, 3, 4, 5 of the particular program stack position into quarters 1 . . . 5 of the computer (blocks 0, 6, 7 are not loaded).

If the user does not wish to have his first symbolic code line assembled into location 400 but say a lower location like 100, an ORG statement of the form:

ORG 100

must precede the first actual symbolic code line (see page 29 of the LOSS manual).

In this case block 0 of the selected program stack position will contain the first 300 location of the absolute program upon completion of the assembly. This block, however, will not be loaded automatically by the monitor. Usually the programmer places an ORG 400 within (in this case) 300 code lines of the ORG 100. It is to the first code line after the ORG 400 that control would go after the

monitor loaded the program. At this point the programmer might load the block 0 of that program stack position, thus having the entire program in the memory. This, of course, restricts the programmer to always assembling that program on a particular program stack position since the read statement used to load the block 0 remains fixed.

B. Returning to Monitor

If quarters 0 and 6 of the memory have been left undisturbed by the user while executing his program the instruction:

```
JMP RETURN
```

will return control to the monitor (in quarter 0). If the user needs quarter 0 and 6 of the memory they can be saved (say on block 6 and 7 of the program stack position) and then read back in just before the JMP RETURN is executed. If one does not wish to save quarters 0 and 6 the instructions:

```
RDC
0150
JMP 0
```

will reload the monitor completely just as was done manually when operations were begun.

C. Accepting Parameters by the User's Program

Since the LOSS manual discusses the intricacies of accepting parameters on pages 40-42 and 54-60 the discussion here will deal only with accepting the parameters submitted to the monitor at the time the user's program call is made.

Parameters of the form

```
/4
TESTDATA
```

which represent a file together with its book number (one has no significance without the other) are converted, immediately upon being

entered, by the monitor. They are converted to two octal integers; the first block of the file and the number of blocks (tape unit 1) allocated to that file when it was defined (see DEFINE). These two integers are saved on a buffer in such a manner that they can be picked up by the user's program as described below. A user, if he knew the actual starting block and length of the file (say starting block is 423, length is 16), could enter:

```
423
 16
```

at which time the monitor would recognize them simply as octal integers and transfer them directly to the buffer. In either of the above cases, that which is seen by the user's program is the same two octal integers representing a starting block and a file length.

Similarly a parameter entry of the form:

```
/5
INPUTDATA
/6
OUTPUTDATA
```

would provide four octal integers to the user's program.

The entry:

```
/1
VOLTS
3125
```

would provide three octal constants to the user's program.

Accepting these quantities requires use of the coding in quarter 0 of the monitor and also information left in quarter 6 (the buffer) when the user's program is loaded. Therefore the parameters should be accepted, first thing, in the following manner if the user intends to destroy the contents of those two quarters.

The coding:

```
                    JMP GET
                    JMP + 3
                    JMP GETCL
Initialization      JMP -----   <-- Insert tag here to which
                                    control should go after
                    ADD 0           getting all parameters.
                    JMP LSTOP

                    LDA;
                    -----       <-- Tag of location into which
                                    the user wants the value to go.
                    JMP LSTEL
                    3776        <-- Control entry described on
                                    page 58 of the LOSS manual.

                    LDA;
One set of four     -----
instructions for    JMP LSTEL
each parameter      3776
                      .
                      .
                      .
                      .
                      .
                      .
                      .

                    LDA;
                    -----
                    JMP LSTEL
                    3776

Terminator          JMP LSTCL
                      .
                      .
                      .   User's
                      .   program
                      .
                      .
```

will pick up the parameters from the monitor's buffer (entered by the user during the program call) and place them in the location named by the user following the LDA; instructions.

A more detailed and extensive discussion of parameter handling techniques is discussed in the LOSS manual, pages 40-42 and 54-60.

6. Updating the Program Stack Index

As mentioned before the assembling of a program into the program stack in no way enters information into the program stack index. This index is stored on block 155 of the program stack tape (unit 0) and is loaded into quarter 5 of the memory when the monitor is in core.

The index is updated by updating an identical image of it, which is stored as a file on tape unit 1, using the EDIT program. If an image of the program stack index is not presently available on tape unit 1 proceed as follows.

Define a file of length one block using the DEFINE program. Determine its absolute location on tape unit 1 by entering any program number (or name) followed by the book number and a question mark:

```
<PROG>>
31              Any number.
/4              Book number in which file is defined.
?               Question mark.
```

The contents of that book index will be printed out and from this the absolute location of the 1 block file just defined can be determined. With the location of the file determined, block 155 of the program stack can be read into core and written out again on the file just defined. An image of the program stack index is thus saved on unit 1.

Using the EDIT program this image of the program stack index can be displayed on the oscilloscope and altered using the normal EDIT

features. The program stack index as it appears for the basic system's program can be seen on Listing 14. Each entry (representing a program stack position) consists of two parts each terminated by a dollar sign. The first is the program name and the second is the parameter request the user wishes to have printed out by the monitor when the name is entered during the program call.

The last entry is followed by two dollar signs together which act as a terminator for the entire index. Notice that unused program stack positions, below the highest one presently in use, must be indicated by a

.$.$

entry. The number of the stack position and its name are related simply by the position of the name in the program stack index. For instance TYPE is in location 10 and EDIT is in location 16 as shown in Listing 14.

After the image of the program stack has been updated as a text on tape unit 1, the contents of the text can be transferred to the program stack tape on unit 0. This is done by reading the single block text into core (using the left and right switches) and writing it out on block 155 of unit 0, the program stack tape.

7. Non-LOSS Structured Data Tapes

It is intended that the data tape used by LOSS (on unit 1 only) be divided into 10 books of 100 blocks each and each book into files (sometimes called texts). The explanation of the DEFINE program above describes more thoroughly this format.

However, it is not absolutely necessary that the data tape be structured into this book-file format. Since the monitor transfers parameters of the form:

/1	Book number
DATA	File name

```
.$.$
.$.$
.$.$
.$.$
.$.$
.$.$
.$.$

<10>
TYPE$(MSG,FMT,FILE)'S$
BLINK$FILE$
DISTAPE$NIL$
DISPLAY$NIL$
DEFINE$1 STRING$
MNTR$.$
EDIT$TEXT$
.$.$

<20>
.$.$
.$.$
.$.$
QA$.$
LASS$STACK #, FILES$
$$
```

LISTING 14

into simply two octal integers, one can substitute for such sets of parameters the two octal integers themselves. These two integers are (1) the first block of the data area on the tape, and (2) the length in blocks of the data area. See "Accepting Parameters" under LASS described above for a more complete description.

APPENDIX A

Since a teletype unit provides a simple, convenient and inexpensive input/output device, it seems to have been universally used on the LINC. A small amount of interface wiring suffices to make the connection to the LINC. Unfortunately, a number of different configurations have evolved. One configuration is described in Information Bulletin #6, issued May 26, 1964 by the Massachusetts Institute of Technology. The address given for inquiries was:

S. M. Ornstein
CENTER DEVELOPMENT OFFICE
for Computer Technology in the Biomedical Sciences
292 Main Street
Cambridge 42, Massachusetts

Phone: 491-1934

The Bulletin also gives a program routine for the configuration, which will be called the "LINC Standard" for the remainder of this appendix.

Another configuration was the full buffered input/output where hardware means were used to make the serial-parallel and parallel-serial conversion of the teletype code. There is no other reference to this configuration in this report.

The LINC in our laboratory was connected to a Model 33 Teletype in 1963 before any standard hardware configuration was recognized. This hardware will be referred to as the "IRL connection," for which all our own routines were written. Table 1 (in this appendix) gives a comparison of the IRL connection with the LINC standard.

TABLE 1

A Comparison Between the IRL and LINC Standard Configuration

	IRL	LINC Standard
Output	$\overline{BR_o} = 1$	$BR_o = 1$
Input Sense	XLO	XLO
Send/Receive Mode	"half duplex"	full duplex

The explanation of the Send/Receive Mode is as follows. In the IRL connection the printer is wired to the keyboard. It types what the key indicates, regardless of whether or not the LINC is monitoring the teletype output. Thus the printer can be operated by either the programmer or the LINC but not both at the same time. This is a variation of the usual half duplex mode, in which the printer at station 1 is activated only by the transmitted signal from station 2 under the control of a data set which determines the direction of information flow. Our software uses the symbol ";" to carriage return and space at the end of a program line. If the operator expects a response from the LINC, he must not touch the keyboard until the response is completed. If the operator is typing when the LINC responds, the message will be garbled. The running of the program, however, will be unaffected. In the full duplex mode of the LINC Standard, the keyboard and the printer operate independently. The program is typed out only if the LINC is instructed to "echo back." In this mode, the teletype may send and receive simultaneously.

From Table 1, it would seem that our programs may be used on those machines which employ the LINC Standard by simply changing all instructions to load BR_o to do just the complement. This is indeed the case. As an alternative, we have developed a simple circuit which enables the user to select the IRL connection or the LINC Standard

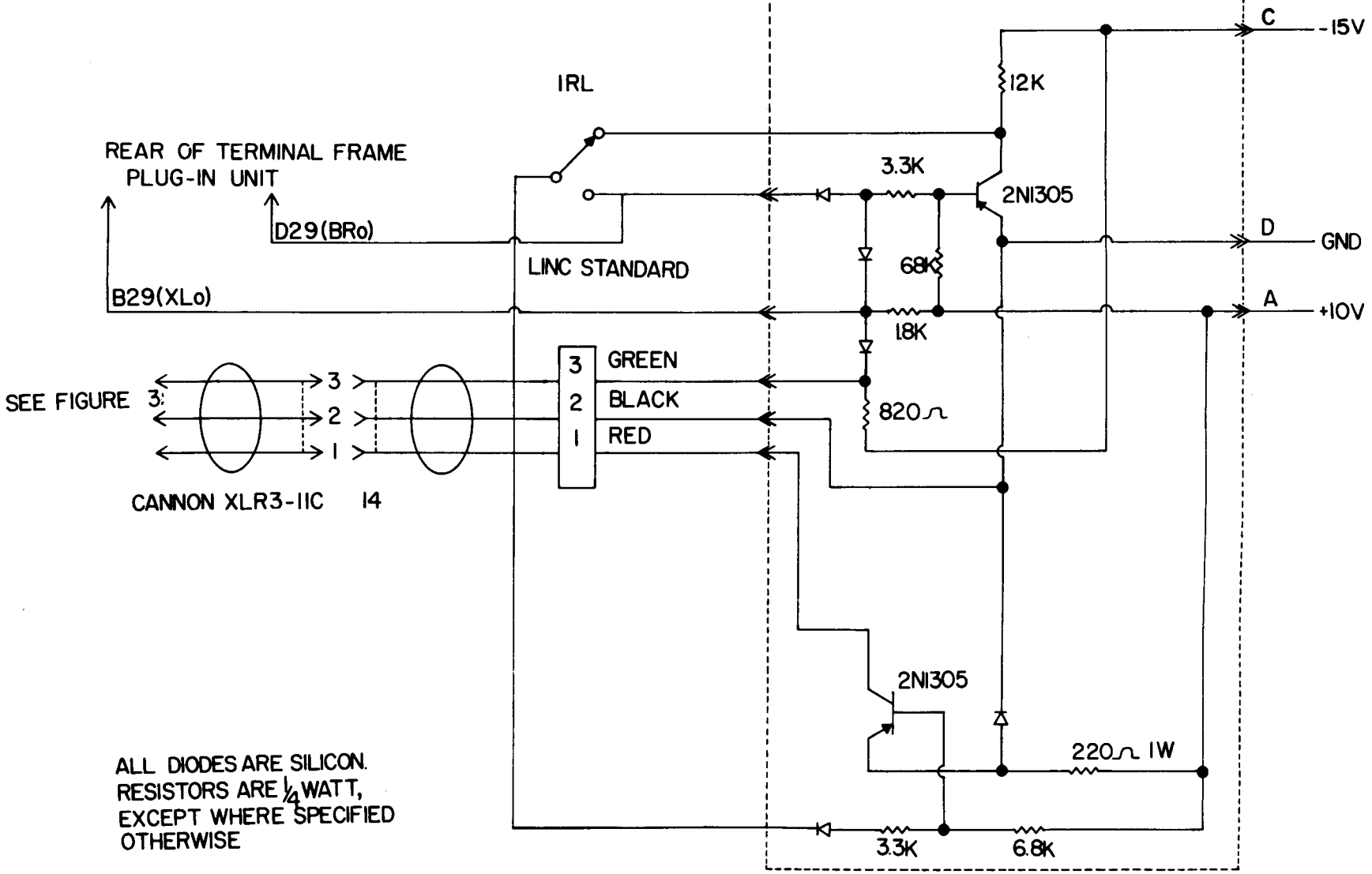
by a single throw of a toggle switch mounted on the front panel of a terminal frame plug-in unit. Thus either software may be used. This circuit is shown in Figure 1, mounted on a blank DEC card of the 4000 series. Figure 2 is a block diagram of the circuit.

Those unfamiliar with the teletype who are planning the hookup would do well to follow the instructions in Bulletin #6 (previously referred to) in ordering the teletype unit. This precaution will minimize the difficulties in making the proper connections. The steps are outlined clearly in the Bulletin, which makes reference to Teletype Drawing 6353 WD. In our case, we discovered that our unit is a stripped down version of the unit described by Drawing 6353 WD. To help those in similar situations, we have drawn an abbreviated schematic (Figure 3) which details the essential connections.

FIGURE 1

CIRCUIT PERMITTING USE OF IRL SOFTWARE OR

LINC STANDARD SOFTWARE BY FLIP OF TOGGLE SWITCH



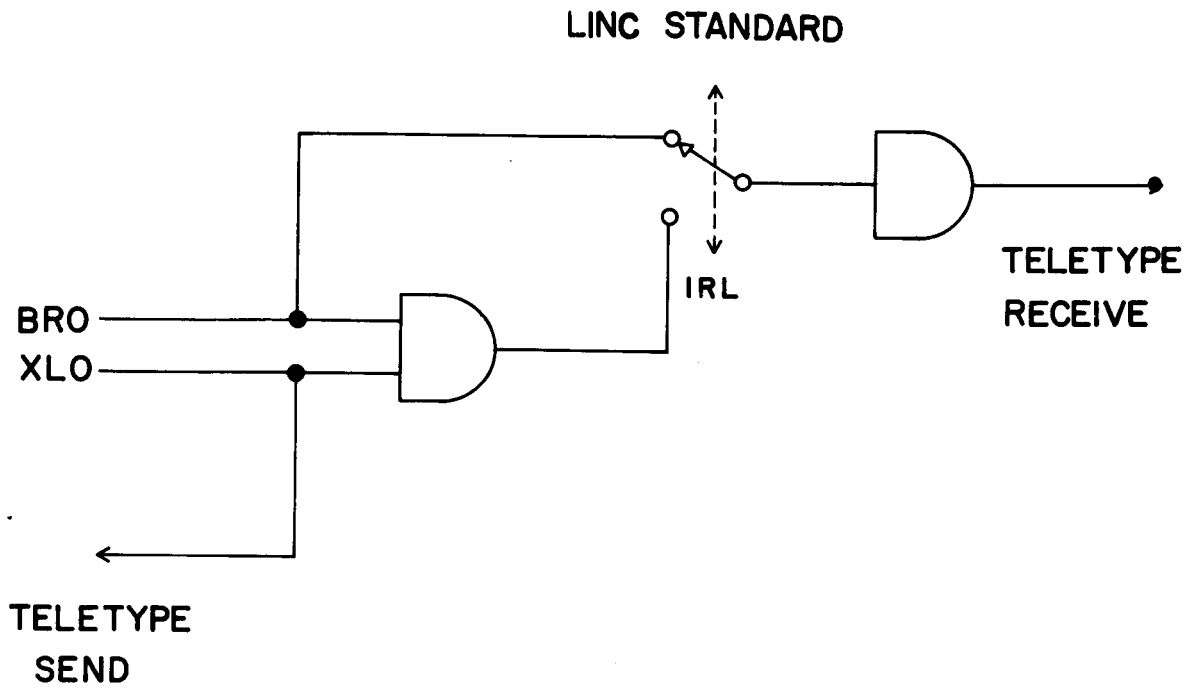


FIGURE 2

A BLOCK DIAGRAM OF FIGURE 1 USING NOR GATES.

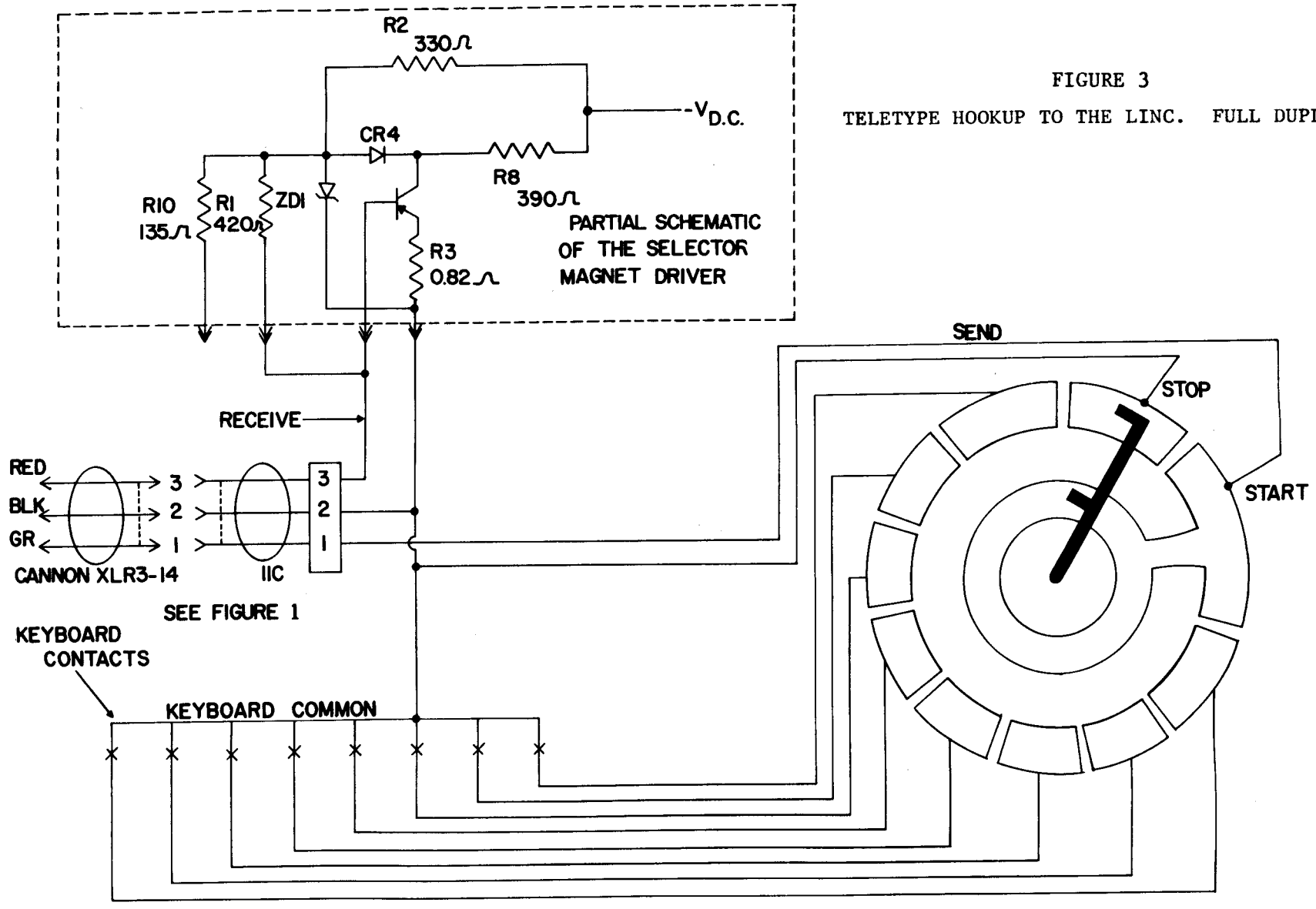


FIGURE 3

TELETYPE HOOKUP TO THE LINC. FULL DUPLEX.

APPENDIX B

The Calcomp Model 565 Digital Incremental Plotter, an extremely useful tool, is easily connected to the LINC. Our own connection, which we term the "IRL",is described in Figure 1 of this appendix. Another connection, which we will call the "LINC Standard," is described in Information Bulletin #4, issued April 27, 1964 by the Massachusetts Institute of Technology. The address given for inquiries was:

S. M. Ornstein
M.I.T. Center Development Office
292 Main Street
Cambridge, Massachusetts 02142

The Calcomp 565 accepts 6 types of commands:

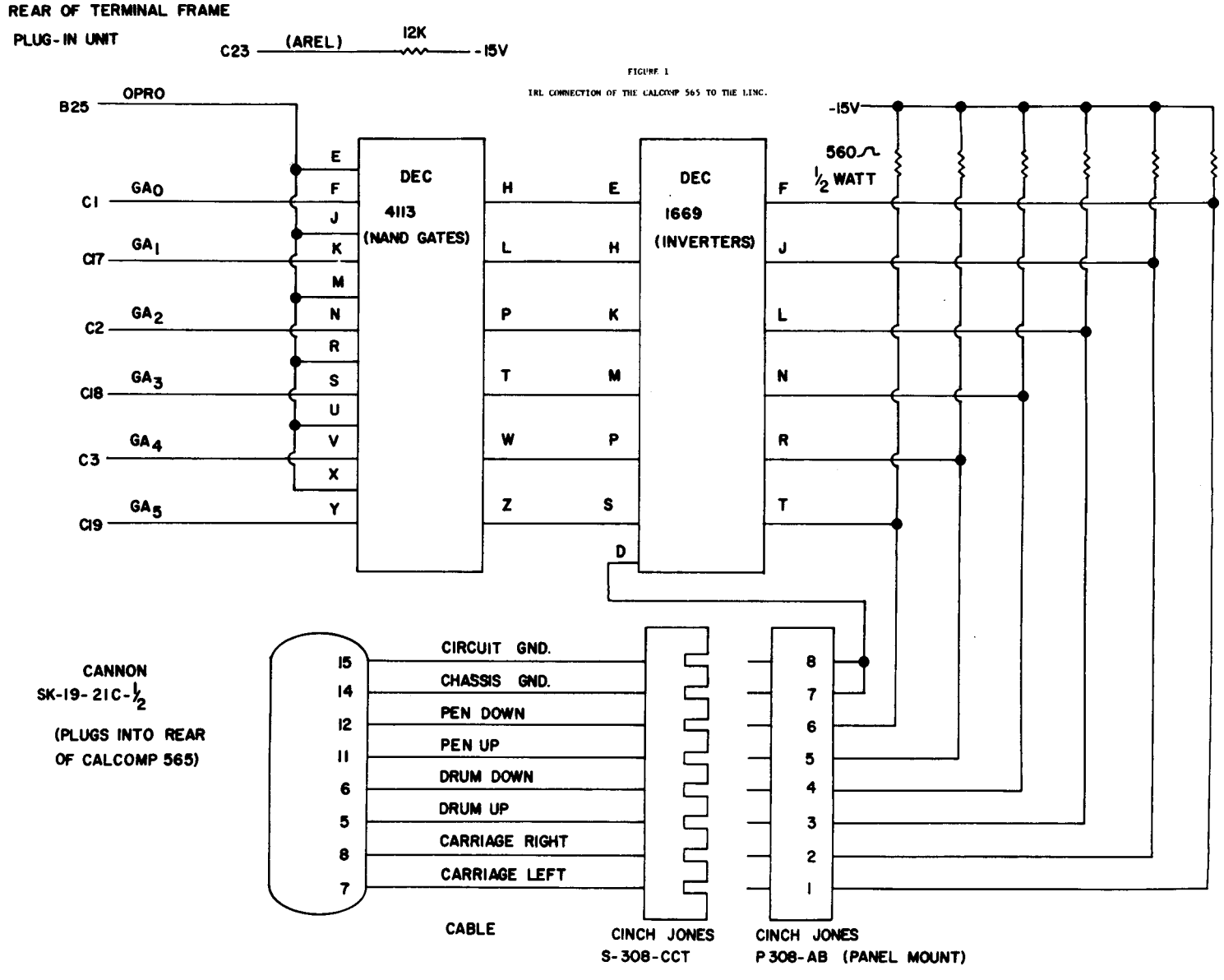
(1) Drum Up
(2) Drum Down
(3) Carriage Right
(4) Carriage Left
(5) Pen Up
(6) Pen Down

The LINC Standard uses a separate OPR line for each of these commands. Our configuration was adopted for the purpose of saving OPR lines for other functions. It also saves a small amount of programming in plotting lines at 45 degrees, although this was not an objective.

To run the Calcomp in the IRL connection, one loads the accumulator with the proper code and executes an OPR 0.

FIGURE 1

IRL CONNECTION OF THE CALCOMP 565 TO THE LINC.



The codes are as follows:

COMMAND	OCTAL CODE
Carriage Left	1
Carriage Right	2
Drum Up	4
Drum Down	10
Pen Up	20
Pen Down	40

BIBLIOGRAPHY

1. Fisher, R. A., Statistical Methods for Research Workers, 11th Ed., Oliver and Boyd (1950).

2. Moore, Richard K., An Operating System for the LINC Computer, NASA Technical Report No. IRL-1038, Instrumentation Research Laboratory, Genetics Department, Stanford University, Palo Alto, California (1965).

3. LINC, Programming and Use I, Vol. 16, Washington University (1965).